



## Материалы для **круглого стола**

«Подготовка специалистов для создания  
современного аппаратного обеспечения  
и программных экосистем»

**Взгляд преподавателя  
педагогического вуза**

# Тезис 1

---

**Обучение по спирали.** В довузовском обучении полезно дать начальное представление об устройстве вычислительной техники (ВТ). Это позволит:

- ▶ сориентировать способных молодых людей в области, связанные с разработкой и производством ВТ;
- ▶ облегчить студентам усвоение материала специализированных технических курсов;
- ▶ получить фундамент для деятельности как в области hardware, так и software.



**профориентация  
+ база**

# Практическое предложение 1

---

Для предварительной подготовки и ориентации абитуриентов на обсуждаемые специальности нужны **популярные образовательные материалы по RISC-V на русском языке.**

Возможно, есть необходимость в **периодическом издании**, где будет соответствующий раздел.

## Тезис 2



**Систематичность и межпредметные связи.** Компьютер – комплексная «многослойная» система (Д. Харрис, С. Харрис – с позиций hardware, Э. Таненбаум – с позиций software). Изучение объекта с разных точек зрения дает **глубину и систематичность** знаний. Еще обеспечивается повторение и **закрепление** материала.

«Областью, в которой "встречаются" программист и конструктор компьютеров, является набор машинных команд».

В. Столлингс

## Практическое предложение 2

---

Система команд RISC-V может быть положена в основу **нескольких курсов** (архитектура, программирование и др.). На ее основе также можно глубже изучить **теоретические вопросы** (например, о двоичном представлении чисел, о *расширении знака*, переполнении и т.д.).

Для этого полезно иметь соответствующие методические материалы.

А теперь несколько слов на основе опыта самостоятельного изучения RISC-V о том, где он, по-моему, может быть *полезен* сообществу.

# Справка по инструкциям

Предлагаю разместить справочные материалы по RISC-V на сайте (возможно, принадлежащем Альянсу). Я вижу это так.

← → ↻ emc.orgfree.com/demo/addi.html ☆ 👤

## Инструкция **ADDI**

ADDI rd, rs1, imm

Набор команд: базовый RV32I

Формат: I

Операнды: rd и rs1 – регистры, imm – 12-битная константа (со знаком!)

Действие: rd := rs1+imm

### Примечания

1. При преобразовании константы к 32-битному формату происходит расширение знака, т.е. знаковый (11-й) бит константы копируется в биты 12-31. (Биты нумеруются справа налево начиная с 0.)
2. Переполнение игнорируется, результатом операции являются младшие биты.
3. Имеется разновидность операции сложения **ADD**, у которой вместо константы imm используется содержимое регистра.
4. Команда ADDI часто используется в псевдооперациях (см. примеры 4-5).

### Пример 1

Инструкция ADDI x31, x7, 6 прибавляет к содержимому регистра x7 число 6, а результат помещает в x31.  
Код инструкции содержит следующие поля:

поле	разрядность	содержимое	примечание
imm	12 битов	000000000110	6
rs1	5 битов	00111	x7
func3	3 бита	000	всегда
rd	5 битов	11111	x31
opcode	7 битов	0010011	всегда

Итоговый код

000000000110 00111 000 11111 0010011<sub>2</sub> = 00 63 8F 93<sub>16</sub>

Пример 2

Рассмотреть подробнее –

<https://emc.orgfree.com/demo/apkit24.html>

Как можно расширить возможности справочной системы?

# Справочное приложение (в работе)

Помимо справки может закодировать набранную инструкцию.  
(В перспективе можно будет и наоборот – по коду восстановить инструкцию).

The screenshot shows a software window titled "RISC-V info". It has two main sections: "Code" and "Decode".

**Code Section:**

- Architecture selection:  RV32I,  RV32C,  RV64I,  RV64C,  RV128I,  RV128C,  RV32M,  RV64M.
- Operation:  pseudo,  rd,  imm.
- Registers:  f3,  i1,  rd,  i2,  op.
- Binary code: 000 0 00111 01100 01
- Assembly: ADDI rd, rd, imm[5..0]
- Sign:  +,  -
- Value: 12
- Set: RV32C, Format: CI

**Decode Section:**

- Input: 03 B1
- Hex: 0000 0000 0000 0000 0000 0011 1011 0001
- Buttons: Copy, Decode
- Count: Всего инструкций - 136

**Instruction Details:**

**Инструкция C.ADDI (+ C.NOP)**  
C.ADDI rd, imm

Набор команд: RV32C  
Формат: CI  
Операнды: rd=rs1 - регистр, imm - 6-битная константа (со знаком!)  
Действие: rd := rd+imm  
Расширяется в команду: ADDI rd, rd, imm[5..0]  
Ограничения: rd ≠ 0 (т.е. не x0), imm ≠ 0  
Варианты: rd = 0, imm ≠ 0 - C.NOP; rd = 0, imm = 0 - HINT  
Quadrant (of the encoding space) - биты 1-й и 0-й: 01  
Примечания

- У константы происходит **расширение знака**, т.е. знаковый (5-й) бит константы копируется во все старшие биты. (Биты нумеруются справа налево начиная с 0.)
- Переполнение игнорируется, результатом операции являются младшие биты.
- Имеется разновидность операции сложения C.ADD, у которой вместо константы imm используется содержимое регистра.

**Пример 1.** Инструкция C.ADDI x31, 6 прибавляет к содержимому регистра x31 число 6. Код инструкции содержит следующие поля:

поле	разрядность	содержимое	примечание
func3	3 бита	000	всегда

Если научились кодировать **одну** команду, то следующий шаг – взять **несколько** команд.

# Получаем бонус: учебный ассемблер (в работе)

The screenshot shows the Educational RISC-V Assembler interface. It includes a source code editor, a label table, a RISC-V code window, and an error diagnostic window. Callouts point to these specific features.

**Errors**: No errors

LABEL	ADDRESS	LINE
MAIN	0	MAIN:ADDI X2, X0, 5 # Line 9
AROUND	40	AROUND:SLT X4, X7, X2 # Line 19
END	72	END:ADD X2, X2, X9 # Line 27
DONE	80	DONE:BEQ X2, X2, DONE # Line 29

00 50 01 13 # Line 9; PC=0  
00 C0 01 93 # Line 10; PC=4  
FF 71 83 93 # Line 11; PC=8  
00 23 E2 33 # Line 12; PC=12  
00 41 F2 B3 # Line 13; PC=16  
00 42 82 B3 # Line 14; PC=20  
02 72 88 63 # Line 15; PC=24  
00 41 A2 33 # Line 16; PC=28  
00 02 04 63 # Line 17; PC=32  
00 02 93 # Line 18; PC=36  
00 03 A2 33 # Line 19; PC=40  
00 03 B3 # Line 20; PC=44  
40 03 B3 # Line 21; PC=48  
04 71 83 # Line 22; PC=52  
06 00 00 # Line 23; PC=56

BEQ X4, X0, AROUND # Line 17  
ADDI X5, X0, 0 # Line 18  
AROUND:SLT X4, X7, X2 # Line 19  
X4, X5 # Line 20  
, X7, X2 # Line 21  
X7, 84 # Line 22  
X0, 96 # Line 23  
, X2, X5 # Line 24  
, # Line 25  
X0, 1 # Line 26  
X2, X2, X9 # Line 27  
X2, 0X20 # Line 28  
BEQ X2, X2, DONE # Line 29

Программа:  
тест  
из книги  
Харрисов

Таблица  
меток

Код  
RISC-V

Препроцессинг: замена  
псевдонимов регистров  
и псевдоопераций,  
«стандартизация» текста

Диагностика  
ошибок

Пользуясь случаем, обращаюсь к присутствующим здесь специалистам за подсказкой.

# ОЖИ1: о соответствии кода и ISA

---

(ОЖИ = Очень Желаемая Информация)

RISC-V позволяет произвольно расширять ISA. Очевидно, что компилятор можно «попросить» сгенерировать код RISC-V для любой конкретной ISA (задать базовый набор + имеющиеся расширения).

А вот как потом при загрузке узнать исходную основу кода?

**Подходит ли этот код для исполнения на данном процессоре с имеющейся у него ISA?**

**Характерный пример**

Инструкция **C.LD** читает 64-битное целое в RV64C.

Точно такой же код имеет инструкция загрузки 32-битного вещественного числа **C.FLW**, входящая в набор RV32FC (там нет 64-битных целых!).

**Надо как-то сравнивать конфигурацию процессора и кода.**

Для определения первой достаточно ввести инструкцию вроде **CRUID** в Intel. А вот как узнать вторую?

# ОЖИ2: о взаимодействии ядер

---

(ОЖИ = Очень Желаемая Информация)

Каким образом взаимодействуют ядра в многоядерном процессоре? Как происходит управление ими?

## Характерный пример

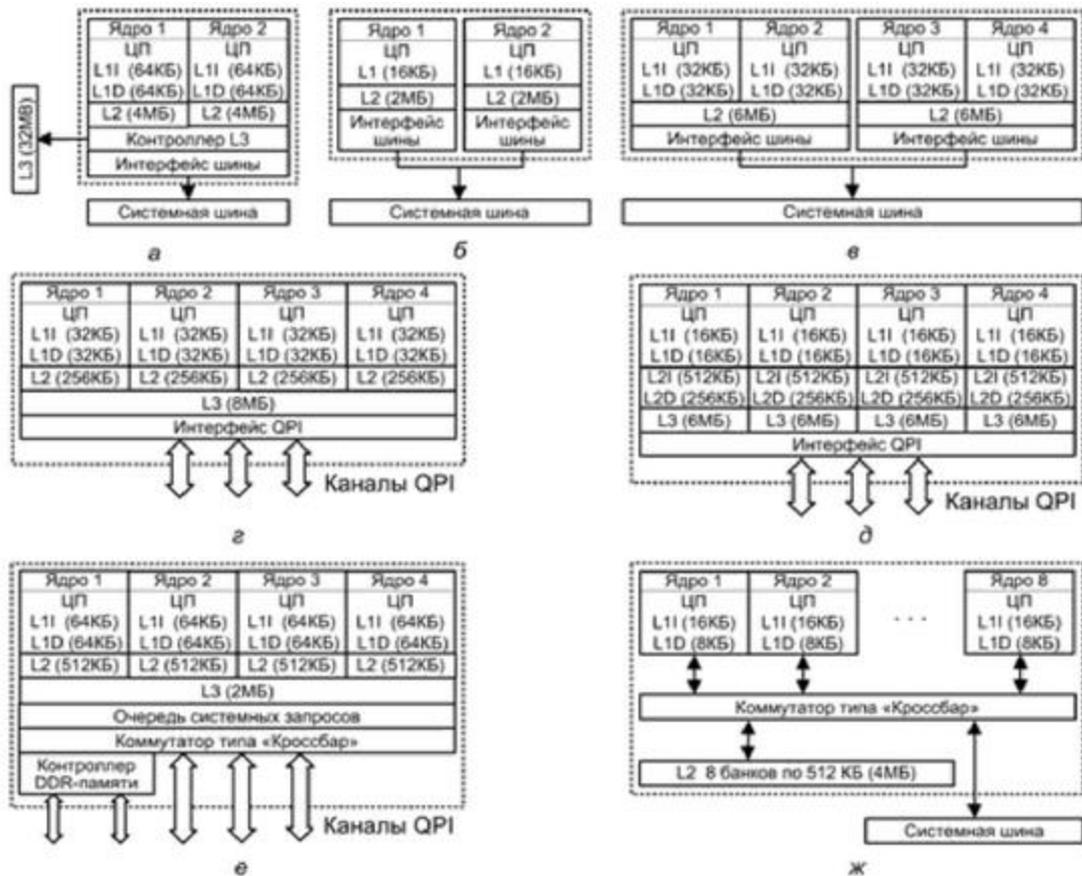
Как стартует система, состоящая из абсолютно одинаковых ядер?

Детали постановки проблемы можно посмотреть в тезисах прошлогодней конференции.

Мне **очень** хочется поработать с многоядерным симулятором!  
А вам?



(дополнение:  
какие они разные...)



**Рис. 9.42.** Структура многоядерных процессоров различных производителей:  
 а — IBM Power 6; б — Intel Pentium D; в — Intel Core 2 Quad; г — Intel Nehalem;  
 д — Intel Itanium 3 Tukwila; е — AMD Phenom X4; ж — Sun UltraSPARC T2

**Орлов С. А., Цилькер Б. Я.**

066 Организация ЭВМ и систем: Учебник для вузов. 2-е изд. — СПб.: Питер, 2011. — 688 с.: ил.

ISBN 978-5-49807-862-5

с. 447