

Содержание

- История появления процессоров Эльбрус
- Модели процессоров Эльбрус
- Архитектура Эльбрус-2000 (e2k), система команд, сравнение с другими архитектурами и семействами архитектур
- Программа первоначального старта
- Средства разработки под процессоры Эльбрус
- Технология Безопасных Вычислений
- Двоичная трансляция (RTC, Lintel)
- Операционные системы
- Прикладное программное обеспечение
- Тесты производительности, игры

История процессоров Эльбрус

История компьютеров Эльбрус. Эльбрус-1

Разработка началась в 1973 в “Институте точной механики и вычислительной техники имени Лебедева” (**ИТМиВТ**) под руководством академика Всеволода Сергеевича Бурцева – учёного в области систем управления и теории конструирования универсальных ЭВМ. При создании машины ориентировались на передовые на то время технологии: суперскалярность и многопроцессорность.

Первый Многопроцессорный вычислительный комплекс (МВК) “Эльбрус-1” был сдан в эксплуатацию в 1980 году. Он мог содержать до 10 процессоров и показывал производительность в 12 млн операций в секунду. Объём оперативной памяти составлял 64 Мбайт (или 2^{20} машинных слов).

Это была первая ЭВМ в Советском Союзе, построенная на базе ТТЛ-микросхем (это была 133 серия микросхем с логикой ТТЛ (быстродействие 10...15 нс на вентиль)). Память на микросхемах [К565РУЗВ \(NMOP\)](#) ёмкостью 16Кбит.

Аппаратура коррекции и контроля позволяет исправлять одиночные и обнаруживать двойные ошибки при считывании информации. Для этого при записи формируются и добавляются к 72-разрядному информационному слову восемь разрядов кода Хемминга. Таким образом, слово, хранящееся в памяти, содержит 80 разрядов, включая: 8 — управляющая информация (тег), 64 — данные, 8 — контрольные разряды кода Хемминга.

Адресная шина имеет разрядность — 24, т.е. размер физической памяти ограничен 16 мега-словами (по 72 бита), 144 Мб данных вместе с тегами.

История компьютеров Эльбрус. Эльбрус-1

[Советские «Эльбрусы» — обзор архитектуры](#)

Система команд — безадресная, стековая,
используется обратная польская запись



История компьютеров Эльбрус. Эльбрус-2

МВК «Эльбрус-2» — разработан в 1977—1984 годах, сдан в 1985 году. Производительность на 10 процессорах (из них 2 считались резервными) — 125 млн оп/с. Построен на базе ЭСЛ интегральных схем ИС-100, из-за высокой потребляемой мощности требовал мощной системы охлаждения. По словам Бориса Бабаяна, всего было выпущено до 200 машин «Эльбрус-2» с разным числом процессоров.

Процессор:

- Размещён в трёх шкафах
- Система команд — безадресная, стековая, используется обратная польская запись
- Тактовая частота — 20 МГц
- Производительность по смеси Гибсон-3 — 12,5 млн оп/сек

ОЗУ:

- логическая организация — тегированная, страничная (размер страницы — 512 слов)
- физически — до 16 млн слов (24-битная физическая адресация) размером 80 бит (из них 8 контрольных), эквивалентный объём — 144 Мбайт
- построена на микросхемах DRAM ЗУ565РУЗВ (16 К * 1)
- используется трёхуровневый интерлидинг[8]

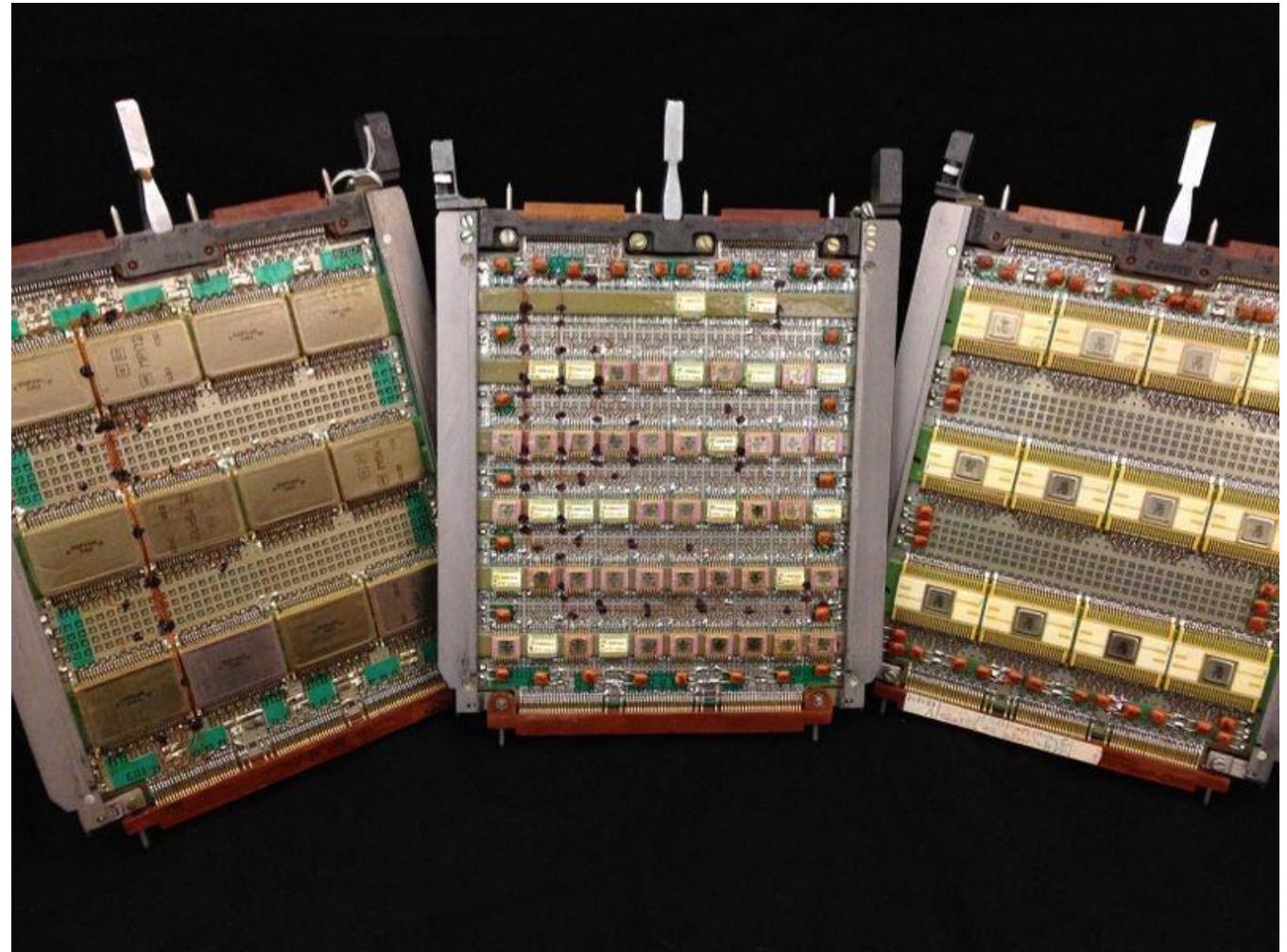
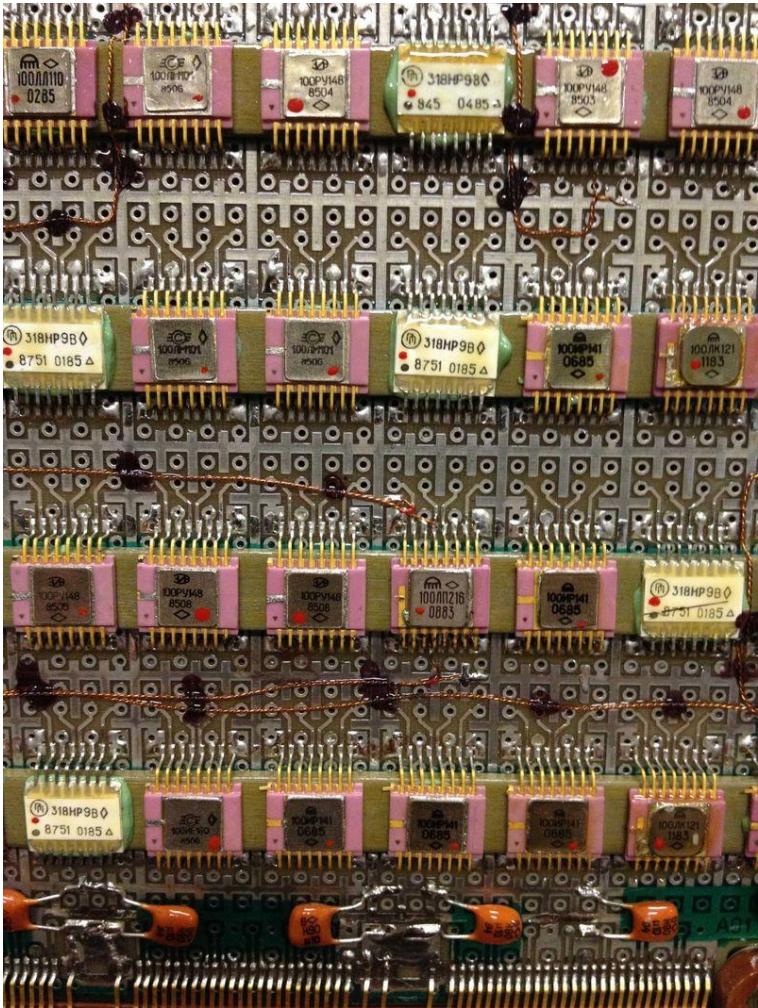
Внешняя память:

- На магнитных барабанах — от 8,5 до 136 Мбайт
- На сменных магнитных дисках — от 34 до 700 Мбайт
- На магнитной ленте — от 70 до 560 Мбайт

История компьютеров Эльбрус. Эльбрус-2



История компьютеров Эльбрус. ТЭЗы.

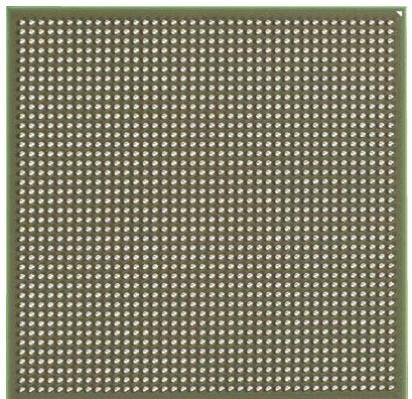


Модели процессоров Эльбрус

Модели процессоров Эльбрус. Таблица.

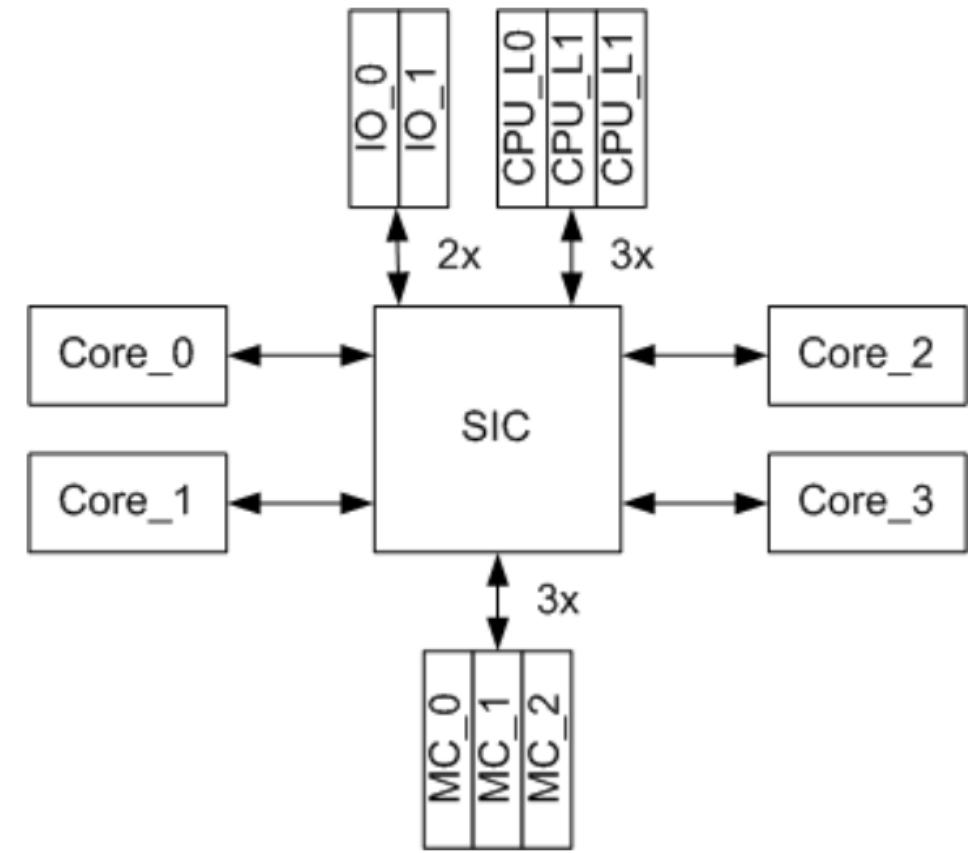
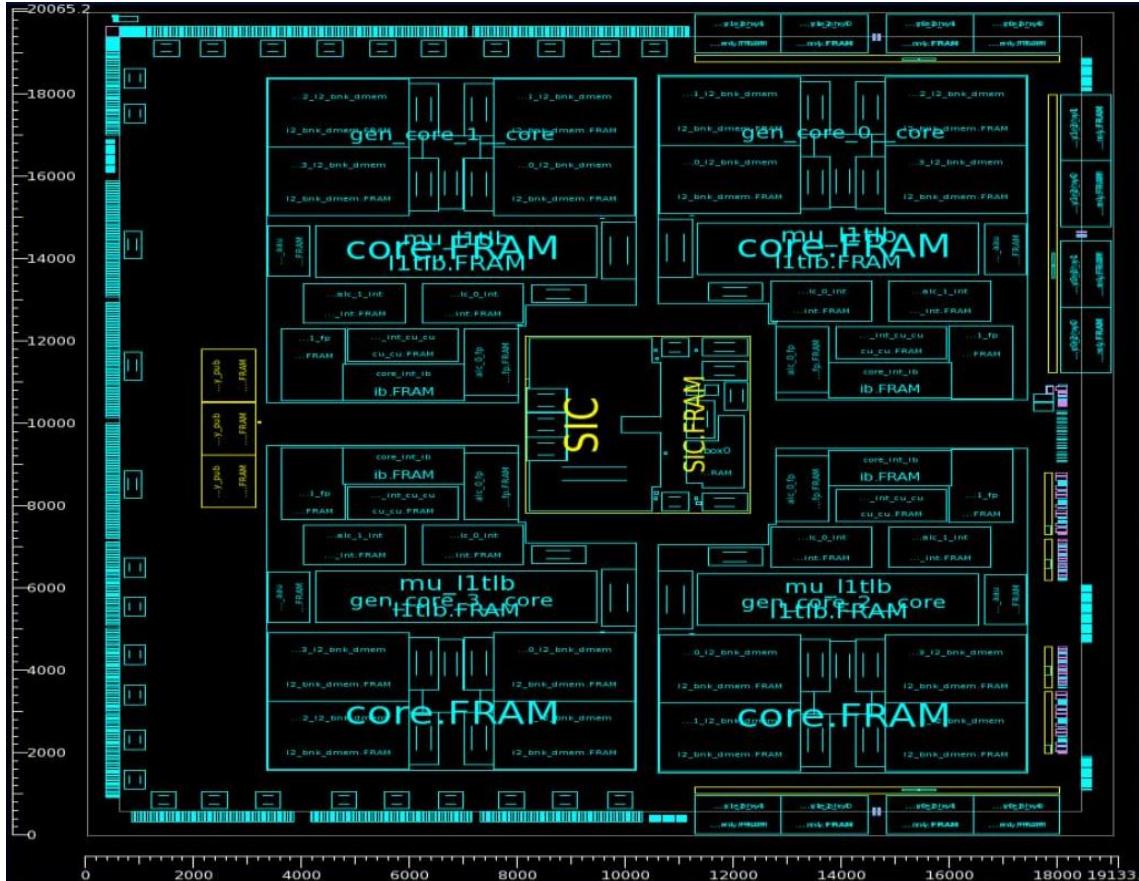
Модель процессора	Год разработки	Микроархитектура, Число команд за такт	Тактовая частота (МГц)	Количество ядер	Кеш	Тип ОЗУ, пропускная способность ОЗУ	Тепловыделение (Вт)	Производительность GFLOP DP	Тех. процесс (нм.), Число транзисторов
Эльбрус-2С+, 1891ВМ7Я	2011	Elbrus-v2; 23 (8 целых, 8 вещественных)	466, 500	2	L1D: 64 Кбайт в ядре, L1I: 64 Кбайт в ядре; L2: 1024 Кбайт в ядре, 2 Мбайт	DDR2-800 ECC, 12,8 Гбайт / с	25	25	90 нм., 368 млн.
Эльбрус-4С, 1891ВМ8Я	2014	Elbrus-v3; 23 (8 целых, 8 вещественных)	700, 750, 800	4	L1D: 64 Кбайт в ядре, L1I: 128 Кбайт в ядре; L2: 2048 Кбайт в ядре, 8 Мбайт суммарно	DDR3-1600 ECC, 38,4 Гбайт / с	60	25	65 нм., 986 млн.
Эльбрус-1С+, 1891ВМ11Я	2016	Elbrus-v4; 25 (8 целых, 12 вещественных), 41 в векторном режиме	600 - 1000	1	L1D: 64 Кбайт в ядре, L1I: 128 Кбайт в ядре; L2: 2048 Кбайт	DDR3-1600 ECC, 25,6 Гбайт / с	10	12	40 нм., 375 млн.
Эльбрус-8С, 1891ВМ10Я, 1891ВМ028	2016	Elbrus-v4; 25 (8 целых, 12 вещественных), 41 в векторном режиме	1000, 1200, 1300	8	L1D: 64 Кбайт в ядре, L1I: 128 Кбайт в ядре; L2: 512 Кбайт в ядре, 4 Мбайт суммарно; L3: 16 Мбайт	DDR3-1600 ECC, 51,2 Гбайт / с	60, 80	125	28 нм., 2,73 млрд.
Эльбрус-8СВ, 1891ВМ12Я, 1891ВМ028	2019	Elbrus-v5; 25 (8 целых, 24 вещественных), 48 в векторном режиме	1200, 1350, 1500	8	L1D: 64 Кбайт в ядре, L1I: 128 Кбайт в ядре; L2: 512 Кбайт в ядре, 4 Мбайт суммарно, L3: 16 Мбайт	DDR4-2400 ECC, 68,3 Гбайт / с	80, 90	288	28 нм., 3,5 млрд.
Эльбрус-2С3, К1891КМ068	2022	Elbrus-v6; 25 (8 целых, 24 вещественных), 48 в векторном режиме	1600, 2000	2	L1D: 64 Кбайт в ядре, L1I: 128 Кбайт в ядре; L2: 2 Мбайт в ядре, 4 Мбайт	DDR4-3200 ECC, 38,4 Гбайт / с	15, 20	96	16 нм., 4,3 млрд.
Эльбрус-16С, К1891ВМ038	2021	Elbrus-v6; 25 (8 целых, 24 вещественных), 48 в векторном режиме	2000	2	L1D: 64 Кбайт в ядре, L1I: 128 Кбайт в ядре; L2: 1 Мбайт в ядре, 16 Мбайт; L3: 32 Мбайт	DDR4-3200 ECC, 200 Гбайт / с	130	768	16 нм., 12 млрд.

Модели процессоров Эльбрус. Эльбрус-4С.



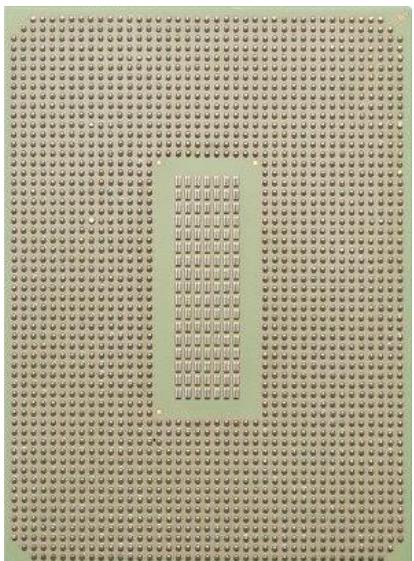
Год выпуска	2014
Архитектура	Эльбрус (Е2К)
Версия системы команд	elbrus-v3
Число ядер	4
Частота	700-800 МГц
Кеш-память	<ul style="list-style-type: none">• L1 D: 64 Кбайт в каждом ядре (Размер линии: 32 байт, Ассоциативность: 4)• L1 I: 128 Кбайт в каждом ядре (Размер линии: 256 байт, Ассоциативность: 4)• L2: 512 Кбайт в каждом ядре, 8 Мбайт суммарно (Размер линии: 64 байт, Ассоциативность: 4)• L3: 16 Мбайт в процессоре (Размер линии: 64 байт, Ассоциативность: 16)
Команд на 1 такт	23 (8 целых, 12 вещественных)
Тип встроенного контроллера памяти	DDR3-1600 ECC
Количество каналов обмена с памятью	3
Пропускная способность шины памяти	38,4 ГБайт / с
Каналы межпроцессорного обмена	3
Южный мост	КПИ
Корпус	HFCBGA 1600
Техпроцесс	65
Число транзисторов	2,73 млрд
Площадь кристалла	380 мм ²
Размеры микросхемы	42,5×42,5×3,2 мм
Масса микросхемы	17 г.

Модели процессоров. Структура Эльбрус-4С.



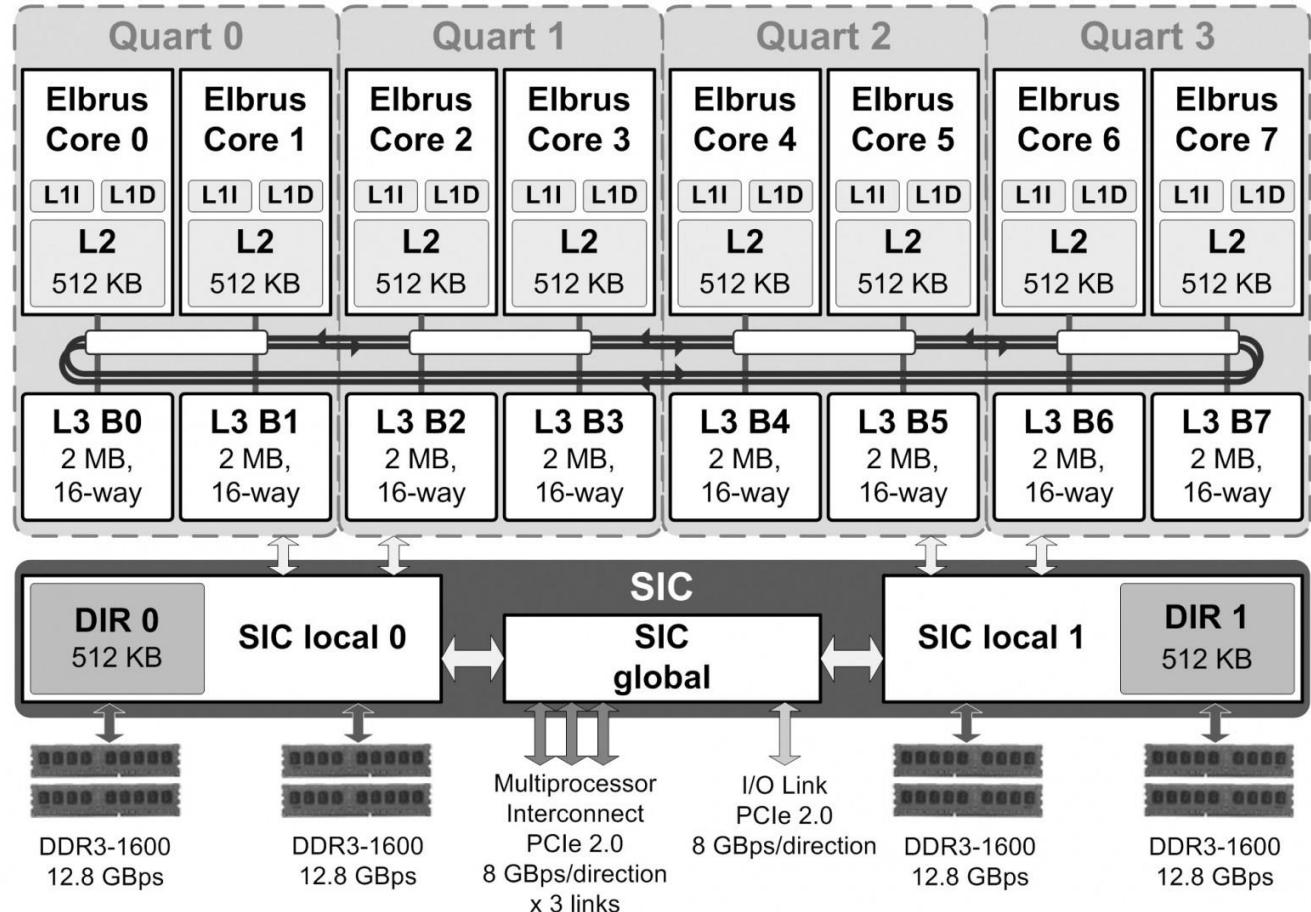
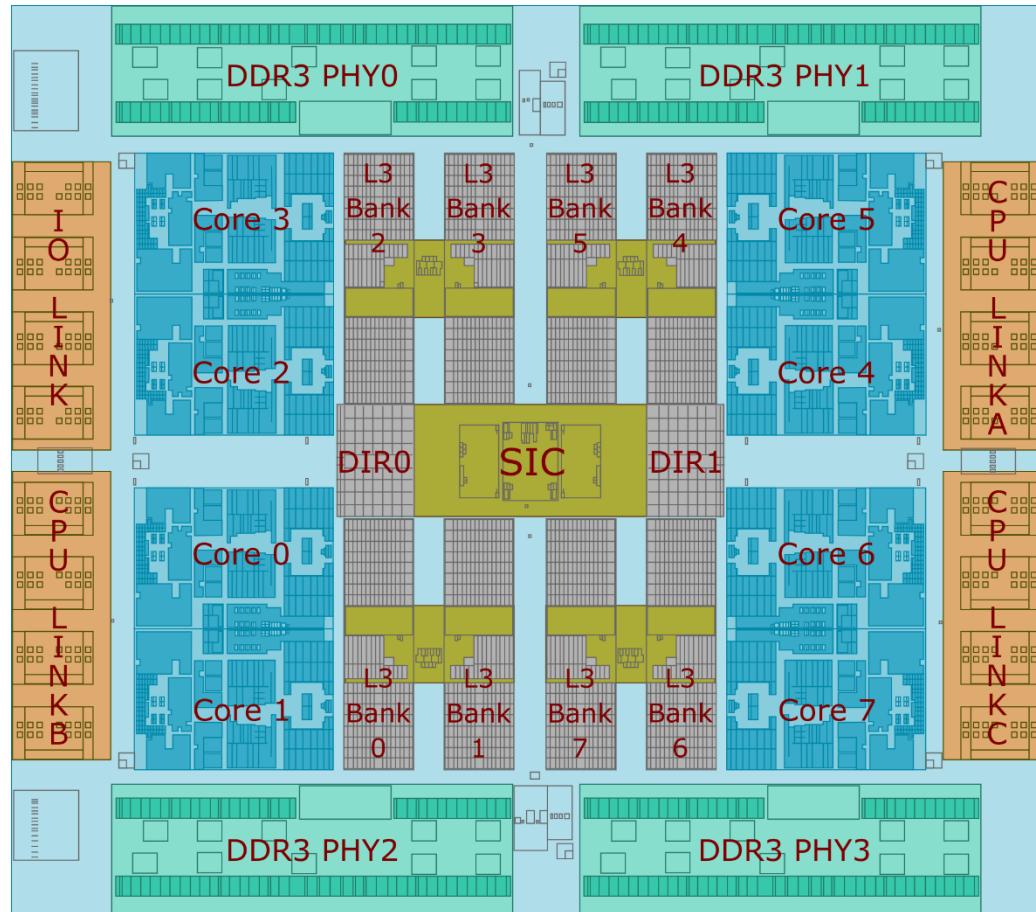
SIC – System Interface Controller

Модели процессоров Эльбрус. Эльбрус-8С.

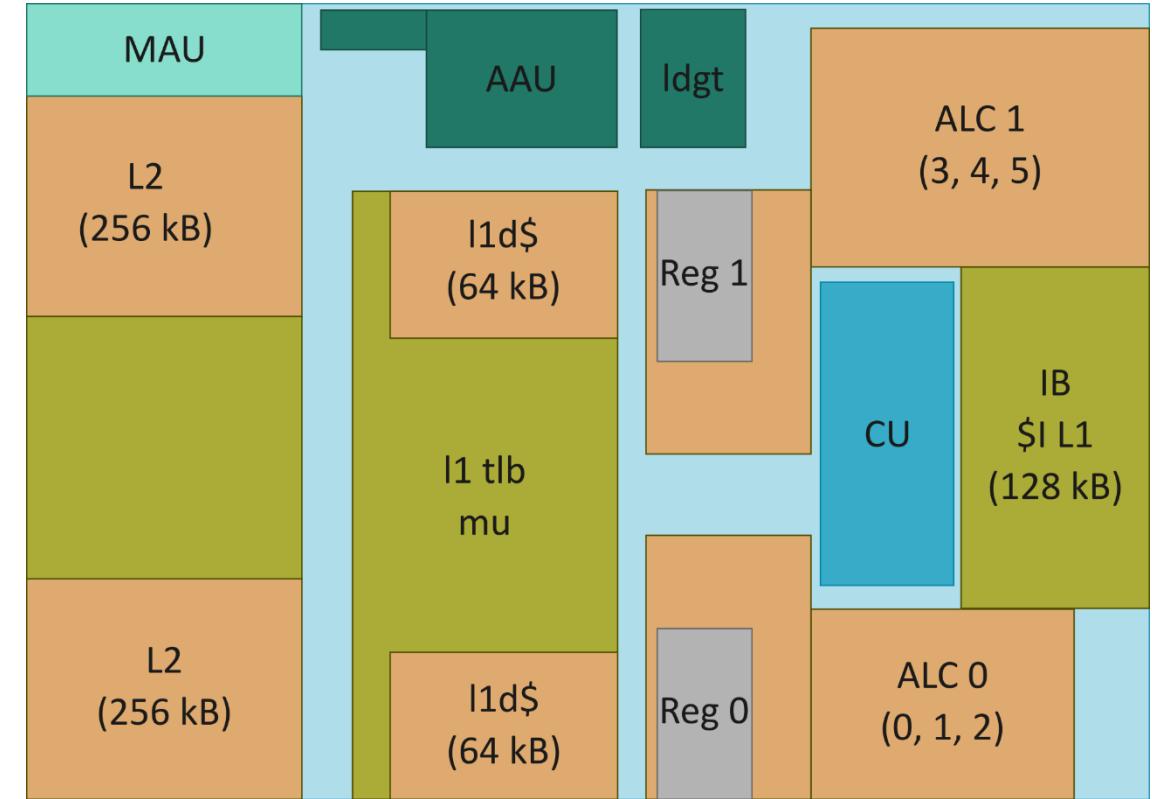
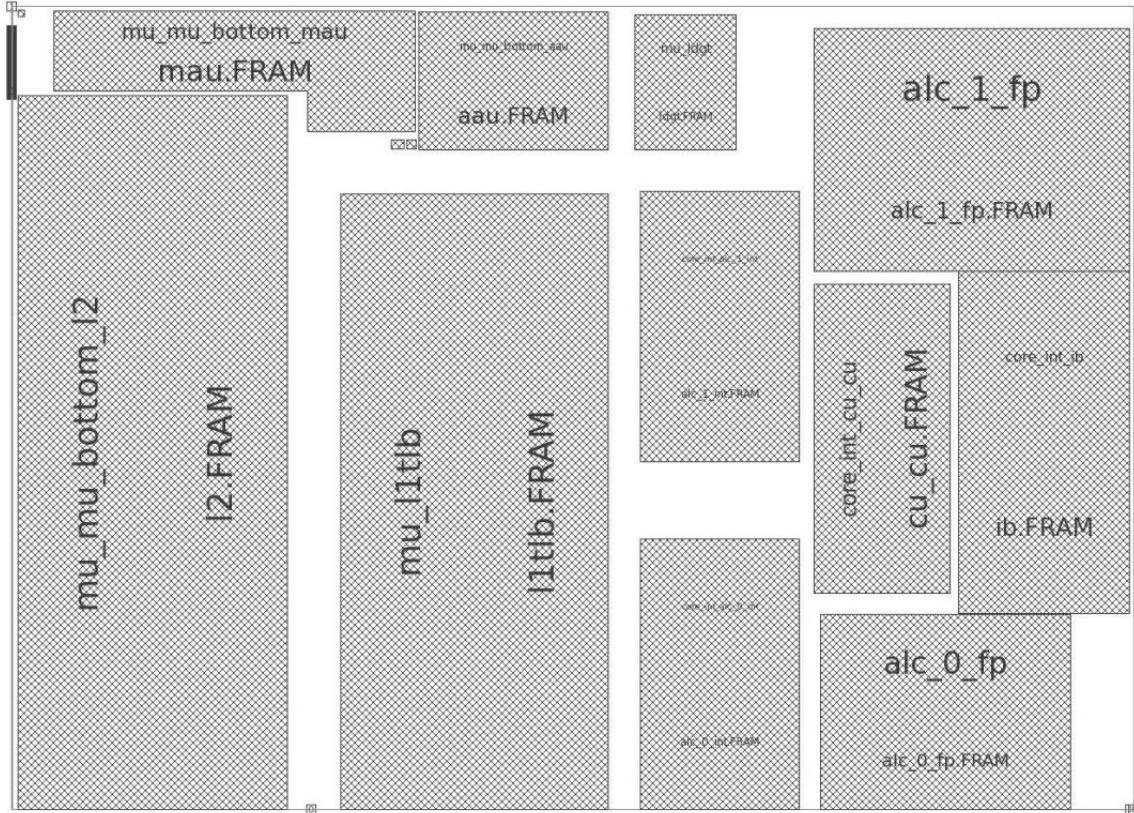


Год выпуска	2015
Архитектура	Эльбрус (Е2К)
Версия системы команд	elbrus-v4
Число ядер	8
Частота	<ul style="list-style-type: none">• 1300 МГц (1891ВМ10АЯ, 1891ВМ02А8)• 1000 МГц (1891ВМ10БЯ, 1891ВМ02Б8)
Кеш-память	<ul style="list-style-type: none">• L1 D: 64 Кбайт в каждом ядре (Размер линии: 32 байт, Ассоциативность: 4)• L1 I: 128 Кбайт в каждом ядре (Размер линии: 256 байт, Ассоциативность: 4)• L2: 512 Кбайт в каждом ядре, 4 Мбайт суммарно (Размер линии: 64 байт, Ассоциативность: 4)• L3: 16 Мбайт в процессоре (Размер линии: 64 байт, Ассоциативность: 16)
Команд на 1 такт	25 (8 целых, 12 вещественных)
Тип встроенного контроллера памяти	DDR3-1600 ECC
Количество каналов обмена с памятью	4
Пропускная способность шины памяти	51,2 Гбайт / с
Каналы межпроцессорного обмена	3
Южный мост	КПИ2
Корпус	FCBGA 2028
Техпроцесс	28 нм
Число транзисторов	2,73 млрд
Площадь кристалла	321 мм ²
Размеры микросхемы	59,5×43,0×4,6 мм
Масса микросхемы	32 г.

Модели процессоров. Структура Эльбрус-8С.



Модели процессоров. Блоки ядра Эльбрус-8С.



MAU – Memory Access Unit

AAU – Array Access Unit

CU – Control unit

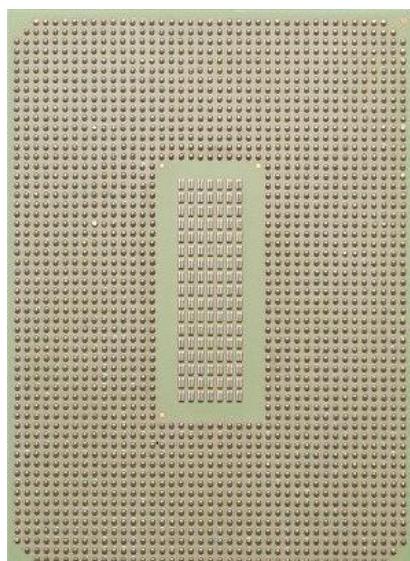
IB – Instruction Buffer

ALC – Arithmetic Logic Channel

TLB – translation lookaside buffer

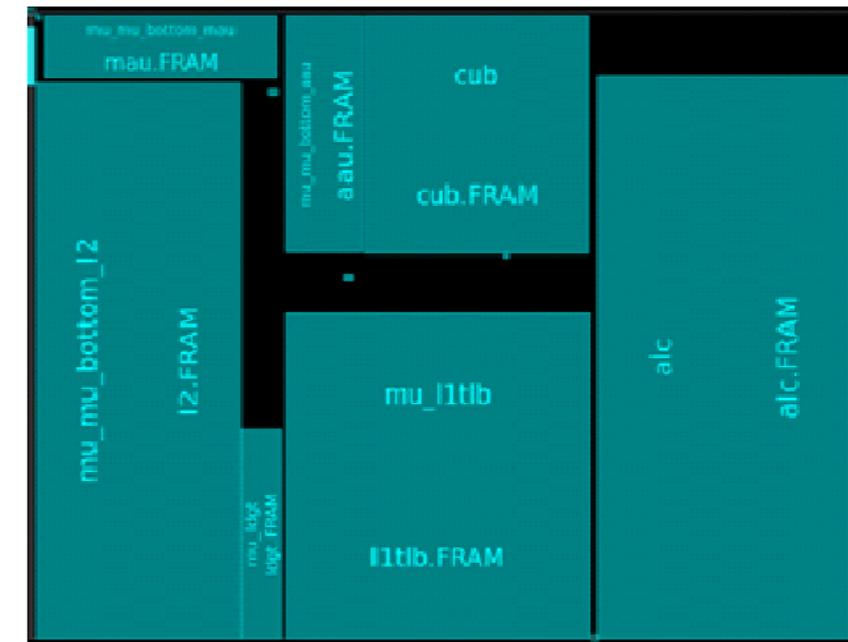
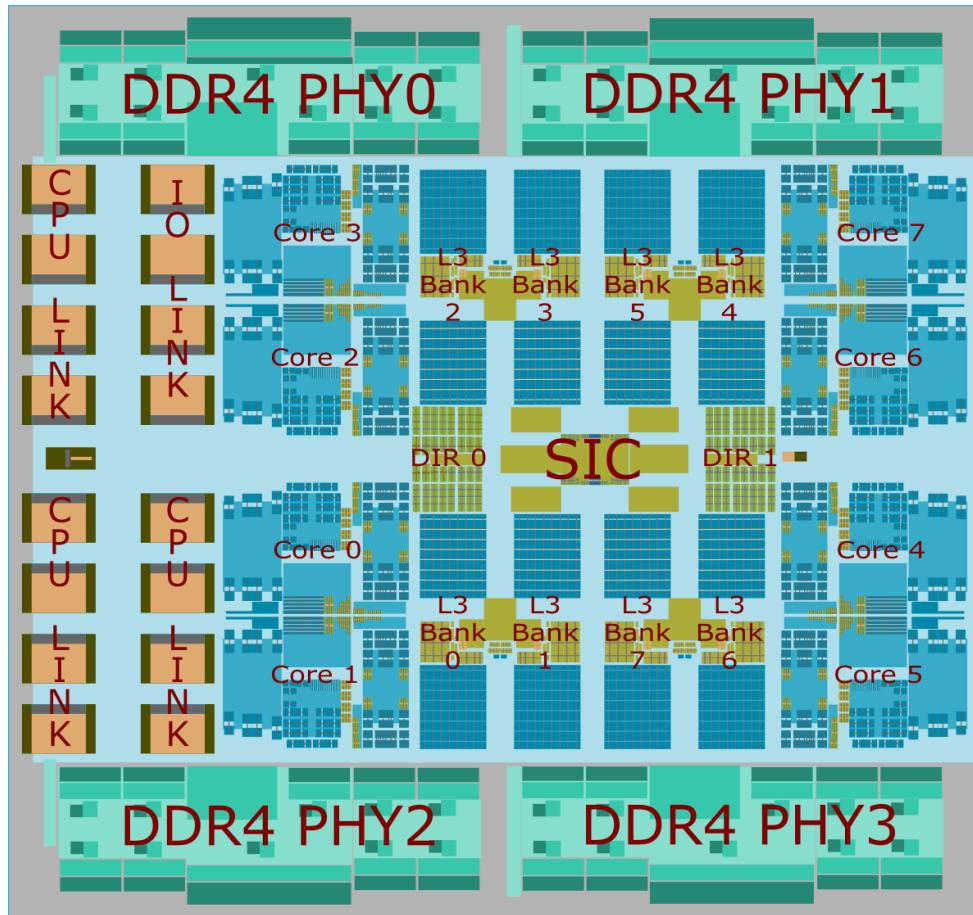
PLU – Predicate Logic Unit

Модели процессоров Эльбрус. Эльбрус-8СВ.

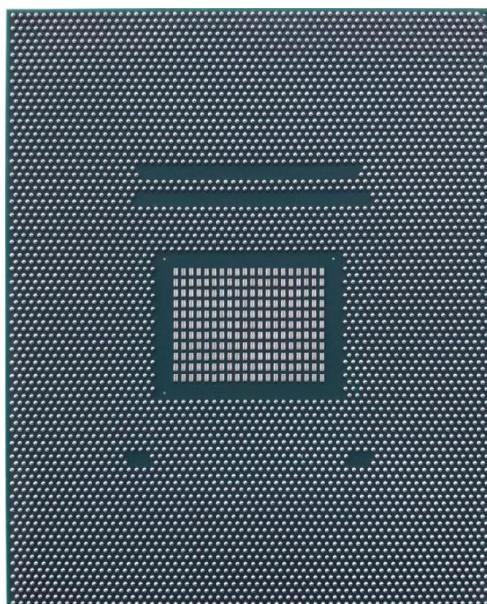


Год выпуска	2018
Архитектура	Эльбрус (E2K)
Версия системы команд	elbrus-v5
Число ядер	8
Частота	1500 МГц (1891ВМ12Я)
Кеш-память	<ul style="list-style-type: none">• L1 D: 64 Кбайт в каждом ядре (Размер линии: 32 байт, Ассоциативность: 4)• L1 I: 128 Кбайт в каждом ядре (Размер линии: 256 байт, Ассоциативность: 4)• L2: 512 Кбайт в каждом ядре, 4 Мбайт суммарно (Размер линии: 64 байт, Ассоциативность: 4)• L3: 16 Мбайт в процессоре (Размер линии: 64 байт, Ассоциативность: 16)
Команд на 1 такт	50 (8 целых, 24 вещественных)
Тип встроенного контроллера памяти	DDR4-2400 ECC
Количество каналов обмена с памятью	4
Пропускная способность шины памяти	68,3 Гбайт / с
Каналы межпроцессорного обмена	3
Южный мост	КПИ2
Корпус	FCBGA 2028
Техпроцесс	28
Число транзисторов	3,5 млрд
Площадь кристалла	350 мм ²
Размеры микросхемы	59,5×43,0×4,6 мм
Масса микросхемы	32 г.

Модели процессоров. Структура Эльбрус-8СВ.

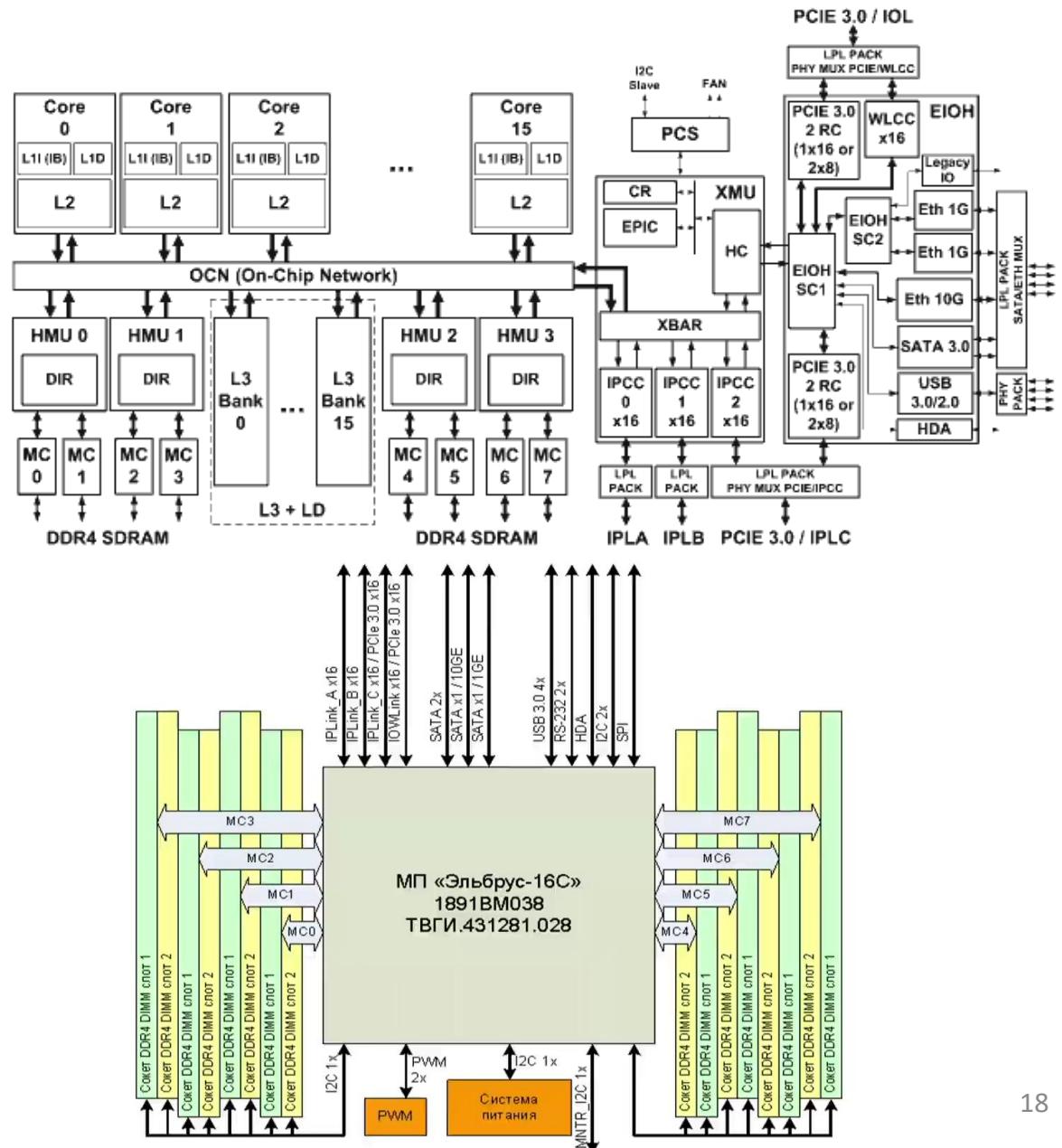
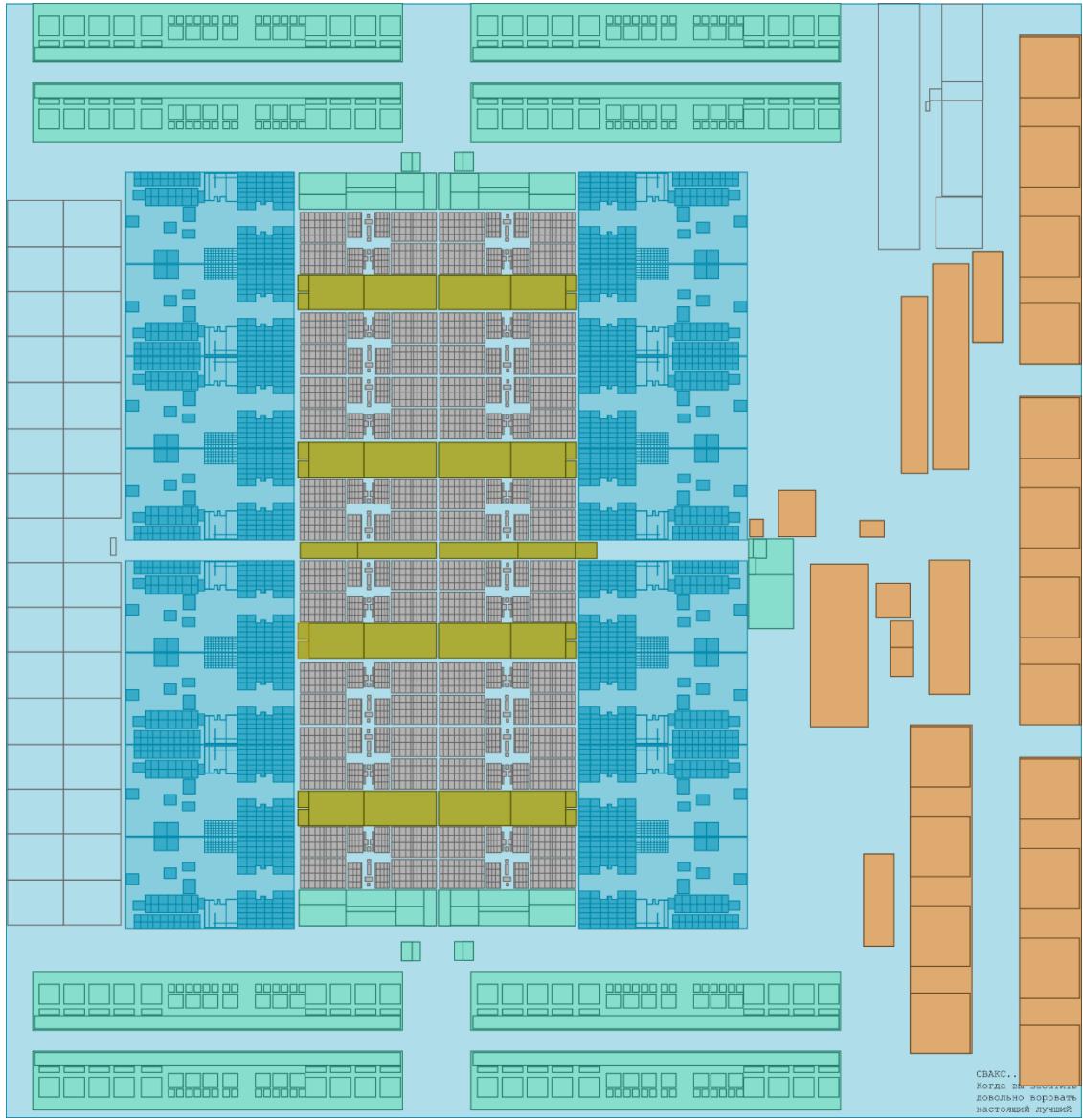


Модели процессоров Эльбрус. Эльбрус-16С.

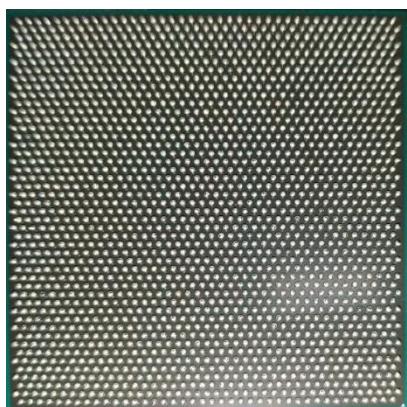


Год выпуска	2021
Архитектура	Эльбрус (E2K)
Версия системы команд	elbrus-v6
Число ядер	16
Частота	до 2000 МГц
Кеш-память	L1 D: 64 Кбайт в каждом ядре (Размер линии: 32 байт, Ассоциативность: 4); L1 I: 128 Кбайт в каждом ядре (Размер линии: 256 байт, Ассоциативность: 4); L2: неинклюзивная, 1024 Кбайт в каждом ядре, 16 Мбайт суммарно (Размер линии: 64 байт, Ассоциативность: 4); L3: неинклюзивная, 32 Мбайт в процессоре (Размер линии: 64 байт, Ассоциативность: 16)
Команд на 1 такт	25-48 (8 целых, 24 вещественных)
Тип встроенного контроллера памяти	DDR4-3200 ECC, поддержка модулей UDIMM, RDIMM, LRDIMM и 3DS
Количество каналов обмена с памятью	8 (поддержка до 16 слотов DIMM, общий объём до 4 ТБ на процессор)
Пропускная способность шины памяти	до 200 Гбайт/с
Каналы межпроцессорного обмена	3 (скорость до 12Gbs; два выделенных шириной до x16, а третий x16/x8/x0 - делится с одним PCIe контроллером)
Южный мост	Интегрированный: PCIe 3.0 (4 контроллера, конфигурации: 2 x16, x16 + 2 x8 или до 4 шт. x8, - совмещаются с межпроцессорными линками и линком к КПИ-2); SATA 3.0 (2 или 4 порта в зависимости от конфигурации); Два Ethernet порта 1Gb/2.5Gb/ 10GbE и 1Gb/2.5Gb (SGMII), совмещены с парой SATA портов; USB 2.0/3.0 (4 порта); Звуковые контроллеры HDA (3 шт.); RS-232 (2 шт.), SPI (4 шт.), I2C (5+1 порт), GPIO (16 линий); Возможность подключение дополнительного южного моста КПИ-2 (за счет линий PCIe).
Корпус	HFCBGA 4804
Техпроцесс	16 нм
TDP	~130 Вт
Число транзисторов, Площадь, Размеры	12 млрд; 618 мм ² (25,3x24,4 мм); 63x78 мм

Модели процессоров. Структура Эльбрус-16С.

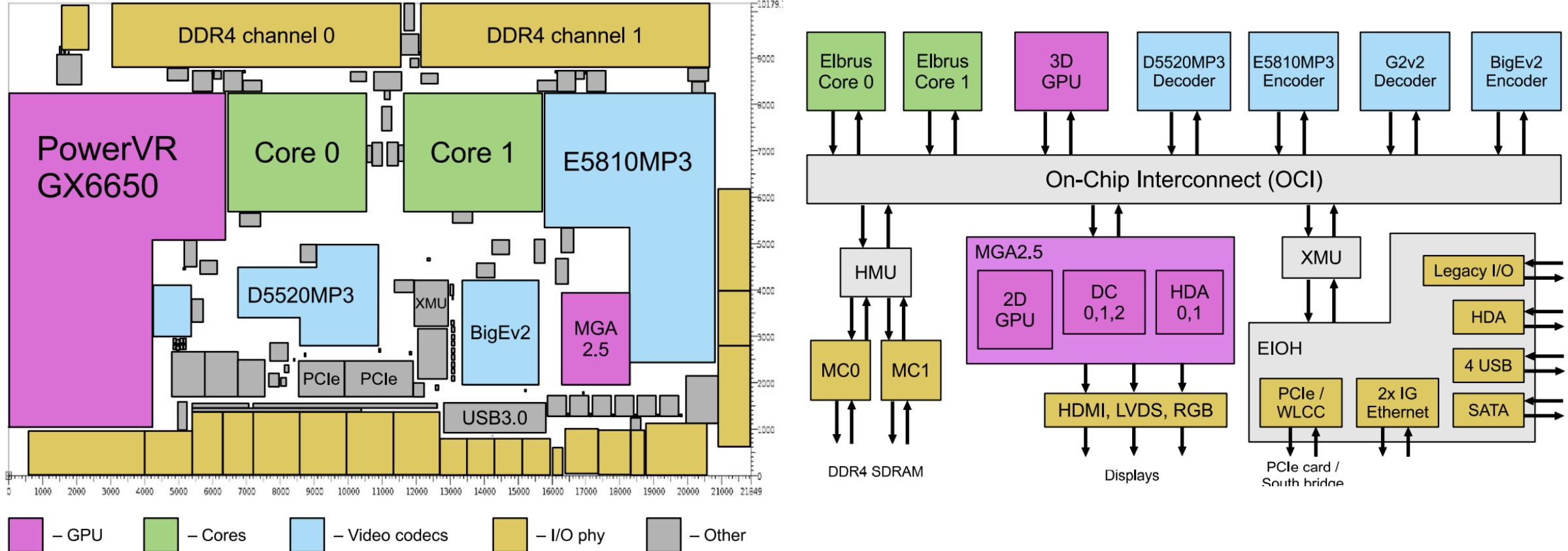


Модели процессоров Эльбрус. Эльбрус-2С3.



Год выпуска	2022
Архитектура	Эльбрус (Е2К)
Версия системы команд	elbrus-v6
Число ядер	2
Частота	до 2000 МГц (K1891KM068)
Кеш-память	<ul style="list-style-type: none">L1 D: 64 Кбайт в каждом ядре (Размер линии: 32 байт, Ассоциативность: 4)L1 I: 128 Кбайт в каждом ядре (Размер линии: 256 байт, Ассоциативность: 4)L2: 2048 Кбайт в каждом ядре, 4 Мбайт суммарно (Размер линии: 64 байт, Ассоциативность: 4)
Команд на 1 такт	25..48
Тип встроенного контроллера памяти	DDR4-3200 ECC
Количество каналов обмена с памятью	2
Пропускная способность шины памяти	51,2 ГБайт / с
Каналы межпроцессорного обмена	-
Южный мост	PCIe 3.0 (4 контроллера общей шириной 0..16 линий); SATA 3.0 (2 или 4 порта в зависимости от конфигурации); Два Ethernet 1Gb/2.5Gb (совмещены с парой SATA портов); USB 2.0/3.0 (4 порта); Звуковые контроллеры HDA (3 шт.); RS-232 (2 шт.), SPI (4 шт.), I2C, GPIO (16 линий); Возможность подключение дополнительного южного моста КПИ-2 (за счет линий PCIe).
Встроенная графика	<ul style="list-style-type: none">2D ядро: МЦСТ МГА2.5 (три независимых дисплейных контроллера, коммутируемые на видеовыходы: два HDMI 4K, LVDS 2.5K, RGB24 FHD, 1000МГц3D ядро: Imagination Series 6XT GX6650 Dastan, 800МГц, до 300 GFLOPSВидеокодеки (работают на частоте 666 МГц): мультистандартный декодер Imagination D5520MP3; энкодер H.264, H.265 Imagination E5810MP3; VP9 Profile 0 (8-bit 4:2:0) декодер (Google G2v2, VP9 Profile 0); VP9 Profile 0 (8-bit 4:2:0) энкодер (Google BigEv2)
Корпус, Техпроцесс	FCBGA 1903, 16 нм
TDP	15 .. 30 Вт макс.

Модели процессоров. Структура Эльбрус-2С3.

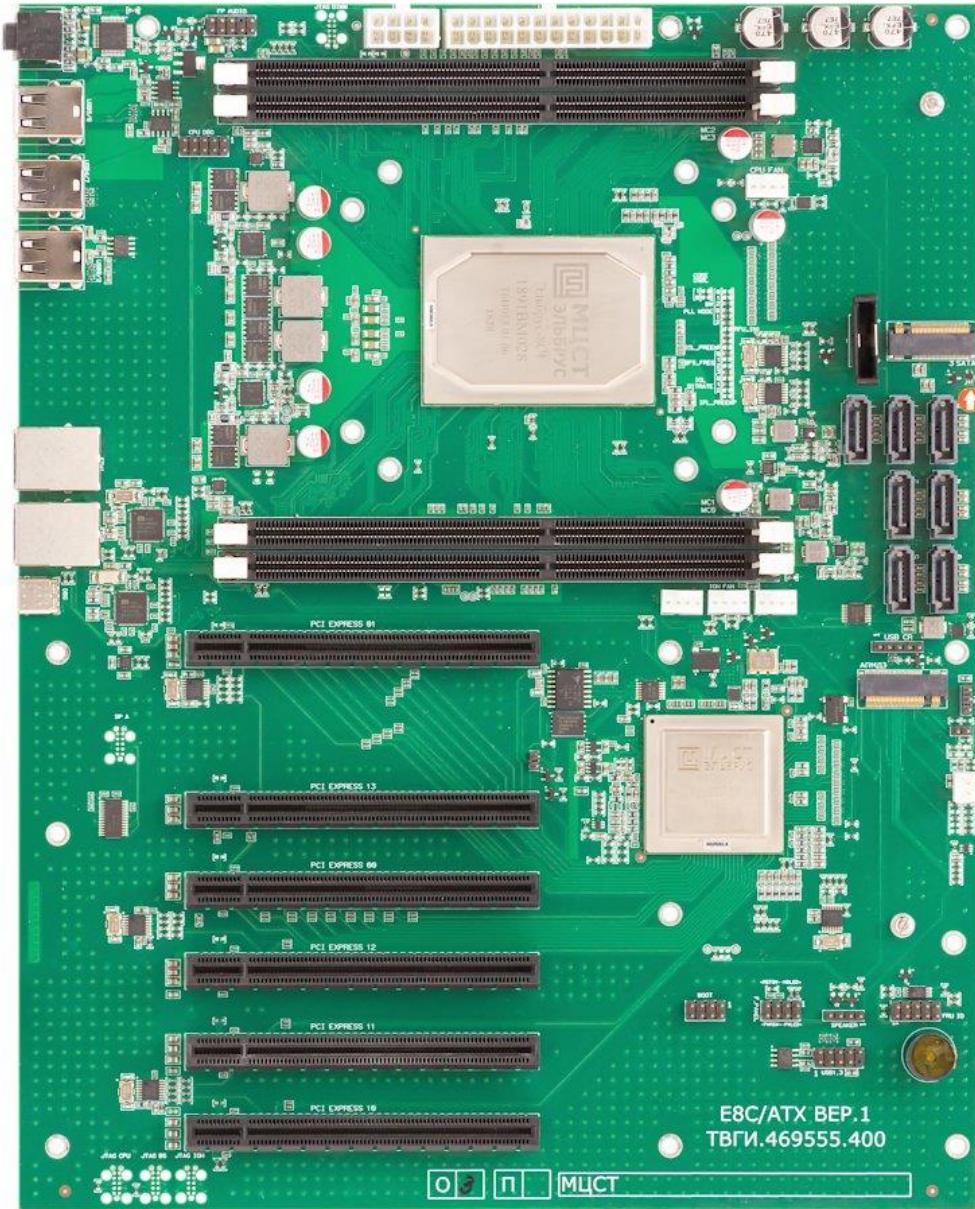
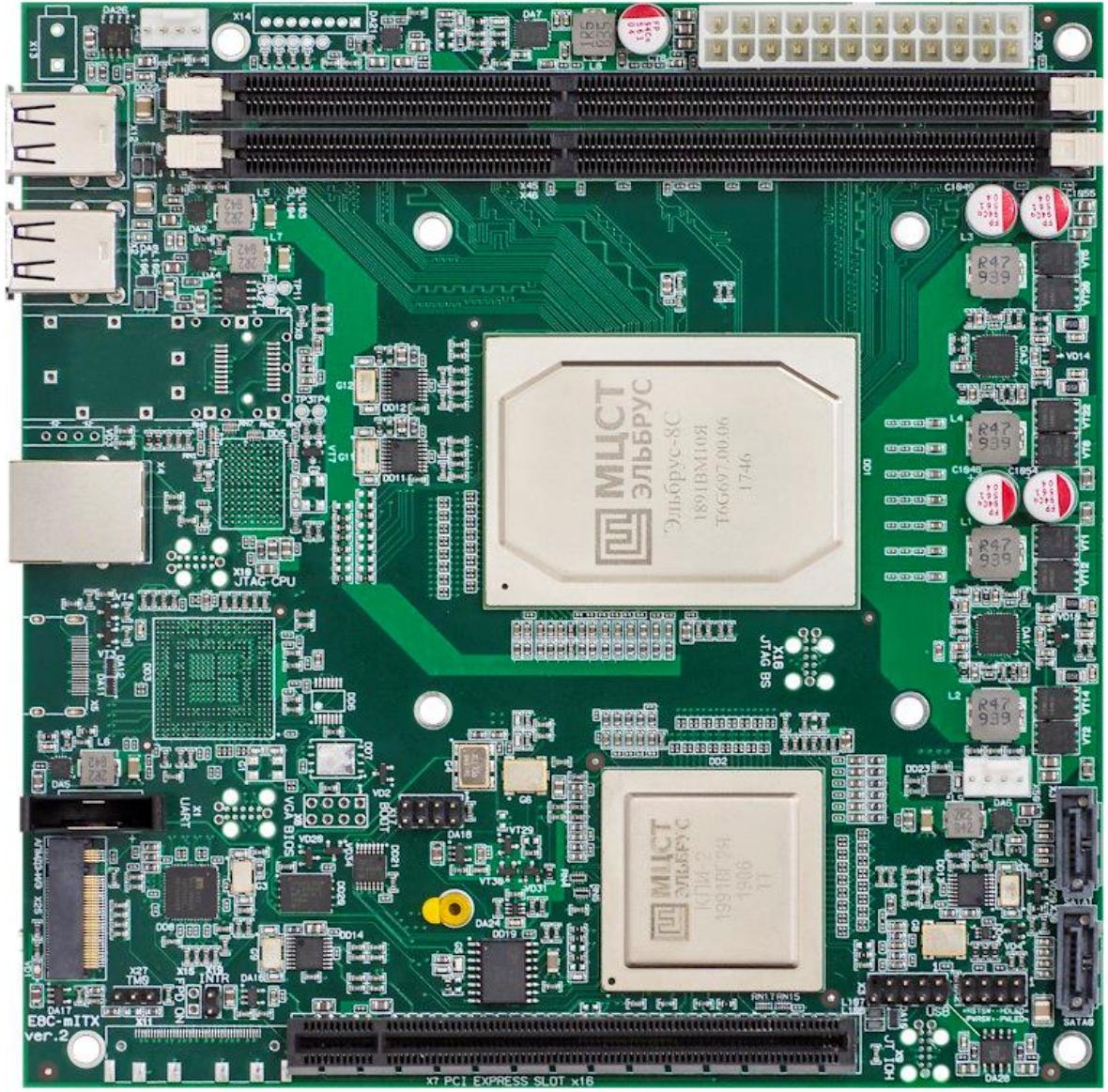


Процессоры Эльбрус. Прототипы FPGA.

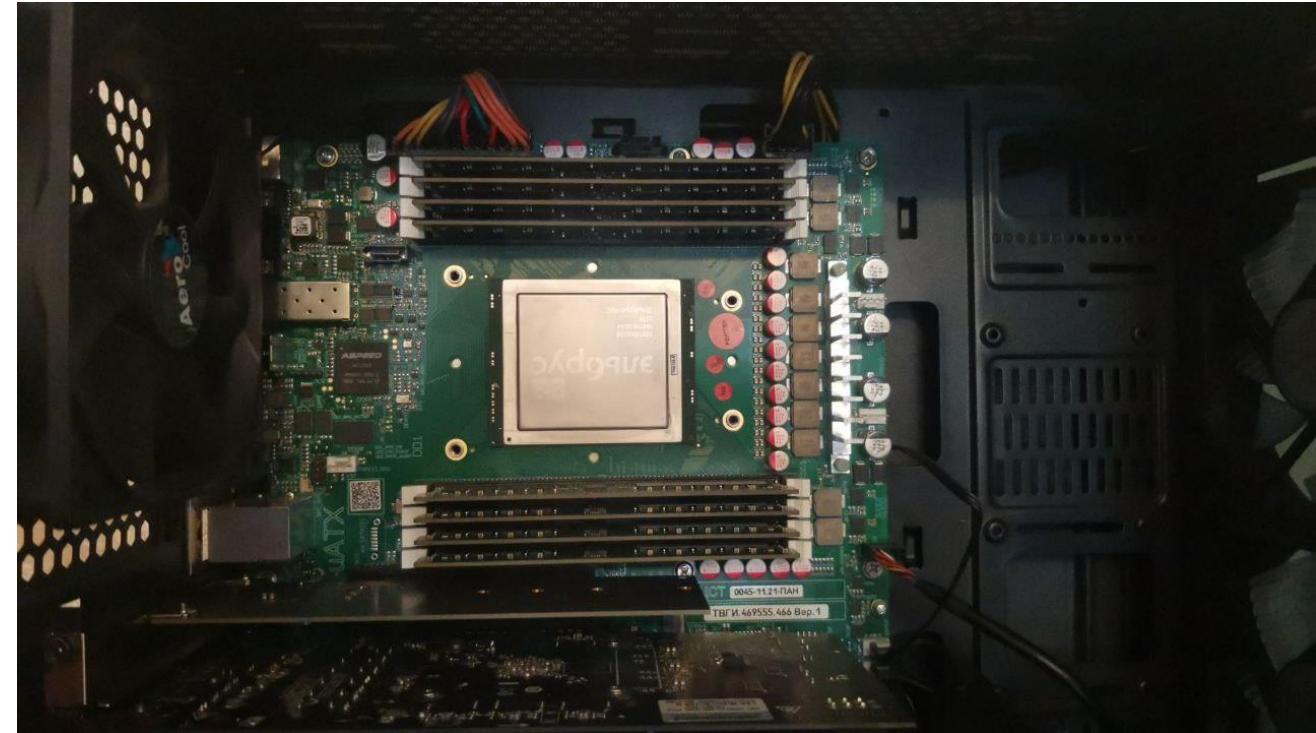
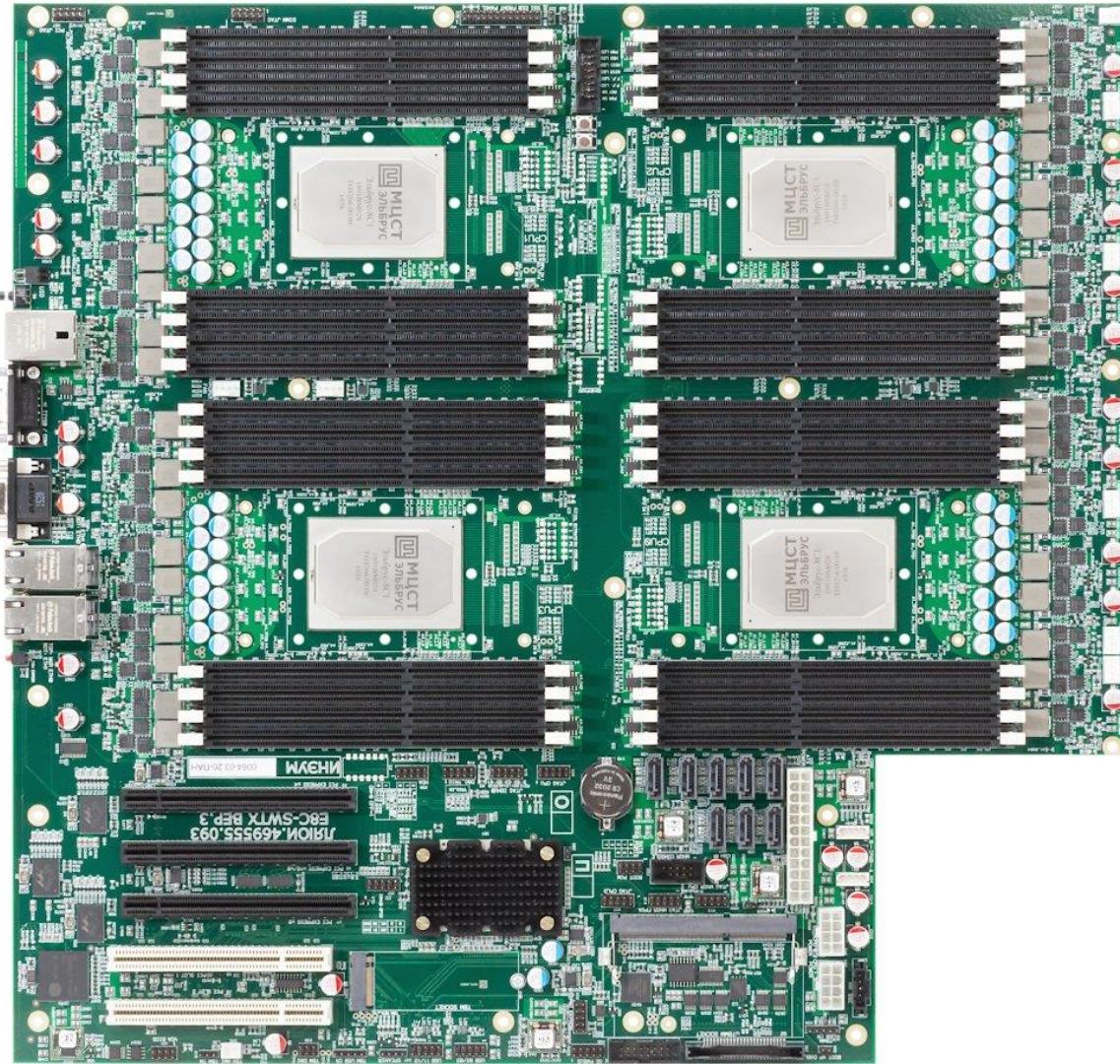
Параметр	Прототип «Эльбрус-4С»	Прототип «Эльбрус-8С»
Количество транзисторов эмулированного микропроцессора, млн. шт.	1112	3000
Количество логических элементов микропроцессора, MGates	~250	~750
Количество процессорных ядер, шт.	4	8
Частота эмуляции, МГц	8,3	9
Используемые ПЛИС	Stratix IV EP4SE820 EP4SE530	Stratix IV EP4SE820
Количество ПЛИС	10	21
Количество доступных логических элементов ПЛИС, млн. шт.	8,1	17
Количество и тип каналов памяти	3 канала DDR3	4 канала DDR3
Размеры корпуса, мм	483x350x289	483x222x716
Стоимость, усл.ед.	1	2
Время изготовления, мес.	6	7
Время ввода в эксплуатацию, мес.	6	7



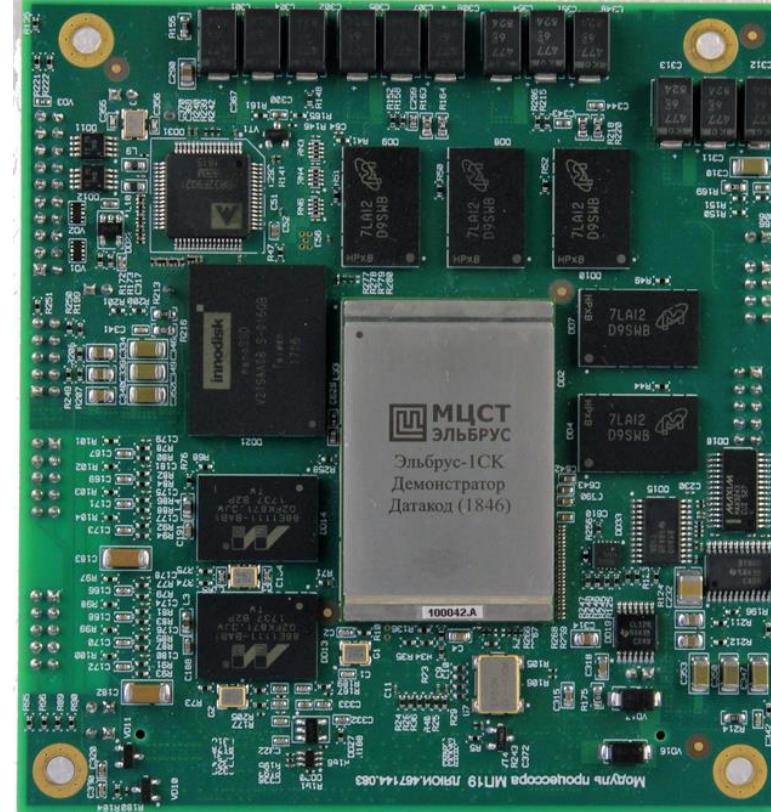
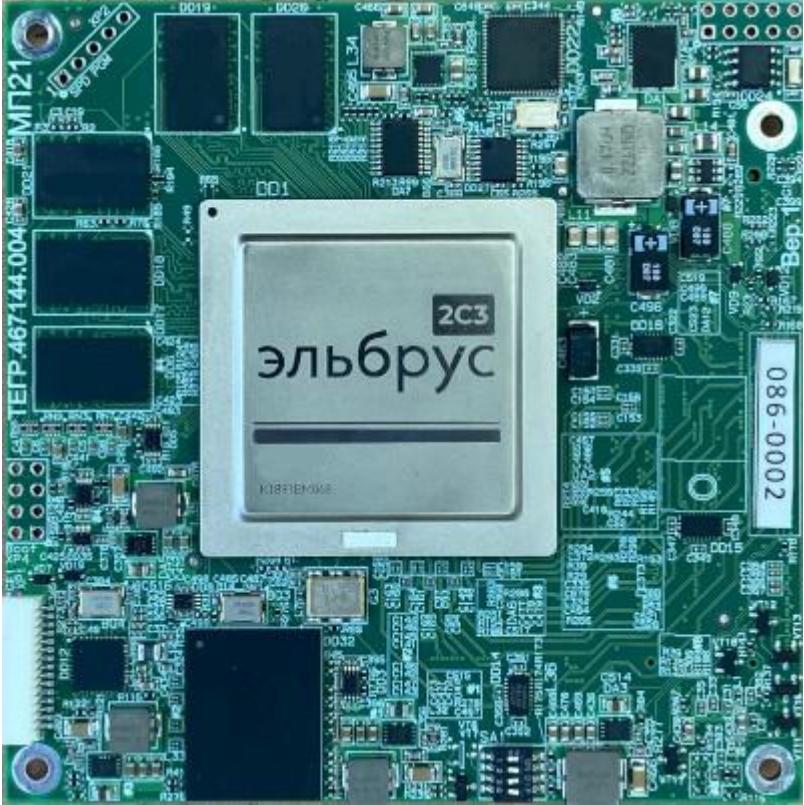
Процессоры Эльбрус. Примеры плат.



Процессоры Эльбрус. Примеры плат.



Процессоры Эльбрус. Примеры плат.



Архитектура процессоров Эльбрус

Архитектура Эльбрус. VLIW

VLIW (Very Long Instruction Word - очень длинная машинная команда) - архитектура процессоров, характеризующаяся возможностью объединения нескольких простых команд в так называемую связку. Входящие в нее команды должны быть независимы друг от друга и выполняться параллельно. Таким образом, из нескольких независимых машинных команд компилятор формирует одно «очень длинное командное слово».

Идея VLIW базируется на том, что задача эффективного планирования параллельного выполнения команд возлагается на сложный компилятор.

Объединение нескольких простых команд в одну сверхдлинную (**Широкая команда, ШК**) производится по следующим правилам:

- количество простых команд, объединяемых в одну команду сверхбольшой длины, равно числу имеющихся в процессоре функциональных (исполнительных) блоков (ФБ);
- в сверхдлинную команду входят только такие простые команды, которые исполняются разными ФБ, то есть обеспечивается одновременное исполнение всех составляющих сверхдлинной команды. Длина сверхдлинной команды обычно составляет от 256 до 1024 битов. Такая метакоманда содержит несколько полей (по числу образующих ее простых команд), каждое из которых описывает операцию для конкретного функционального блока. Длина слова в Эльбрус от **64** до **512** бит.

Архитектурная характеристика. CISC, RISC, VLIW.

Архитектурная Характеристика	CISC	RISC	VLIW
Размер инструкции	Переменный	Фиксированный (32 бит для arm)	Фиксированный (у Эльбруса переменный до 512 бит)
Формат инструкции	Различный (простые и сложные), с префиксами, много зависимых операций в одной	Обычно одна простая команда	Много простых независимых команд
Регистры	Мало, есть специального назначения	Много, общего назначения	Очень много, общего назначения (у Эльбруса к этому есть специального назначения)
Работа с памятью	В составе операций в различных операциях	Не связано с операциями, обычно отдельно загрузить/сохранить	Не связано с операциями, обычно отдельно загрузить/сохранить
Микроархитектура работы с инструкциями	Сложная с микрокодом	Простая с одним конвейером и без микрокода	С несколькими конвейерами и без микрокода, без сложной логики
Графическое Представление (1 клетка – 1 байт)			

Архитектура Эльбрус-2000.

Архитектура «Эльбрус 2000» (e2k) — уникальная российская разработка, которая разработана в конце 90-х — начале 2000-х годов на базе архитектуры «Эльбрус-3». Процессор на базе архитектуры «Эльбрус 2000» оперирует т. н. «широкими командами» (ШК) (VLIW — Very Long Instruction Word) размерностью от **64** до **512** разрядов в которой закодированы команды («слоги») для всех исполнительных устройств. Процессор не анализирует зависимости operandов инструкций и исполняет инструкции последовательно. Это было необходимо для упрощения физического проектирования в кристалле. Процессор работает на достаточно низких тактовых частотах в сравнении с зарубежными аналогами, но использование широких команд позволяет сильно повысить производительность.

Всю работу по глубокому анализу зависимостей кода берёт на себя фирменный компилятор МЦСТ — LCC (eLbrus C Compiler). В некоторых случаях компилятор способен анализировать исходный код гораздо тщательнее, чем аппаратура RISC/CISC процессора, и находить независимые операции в существенно большем диапазоне. Это является основной причиной, почему в архитектуре Эльбрус 2000 больше параллельно работающих исполнительных устройств, чем в традиционных процессорных архитектурах (на момент проектирования архитектуры Эльбрус-2000 было именно так).

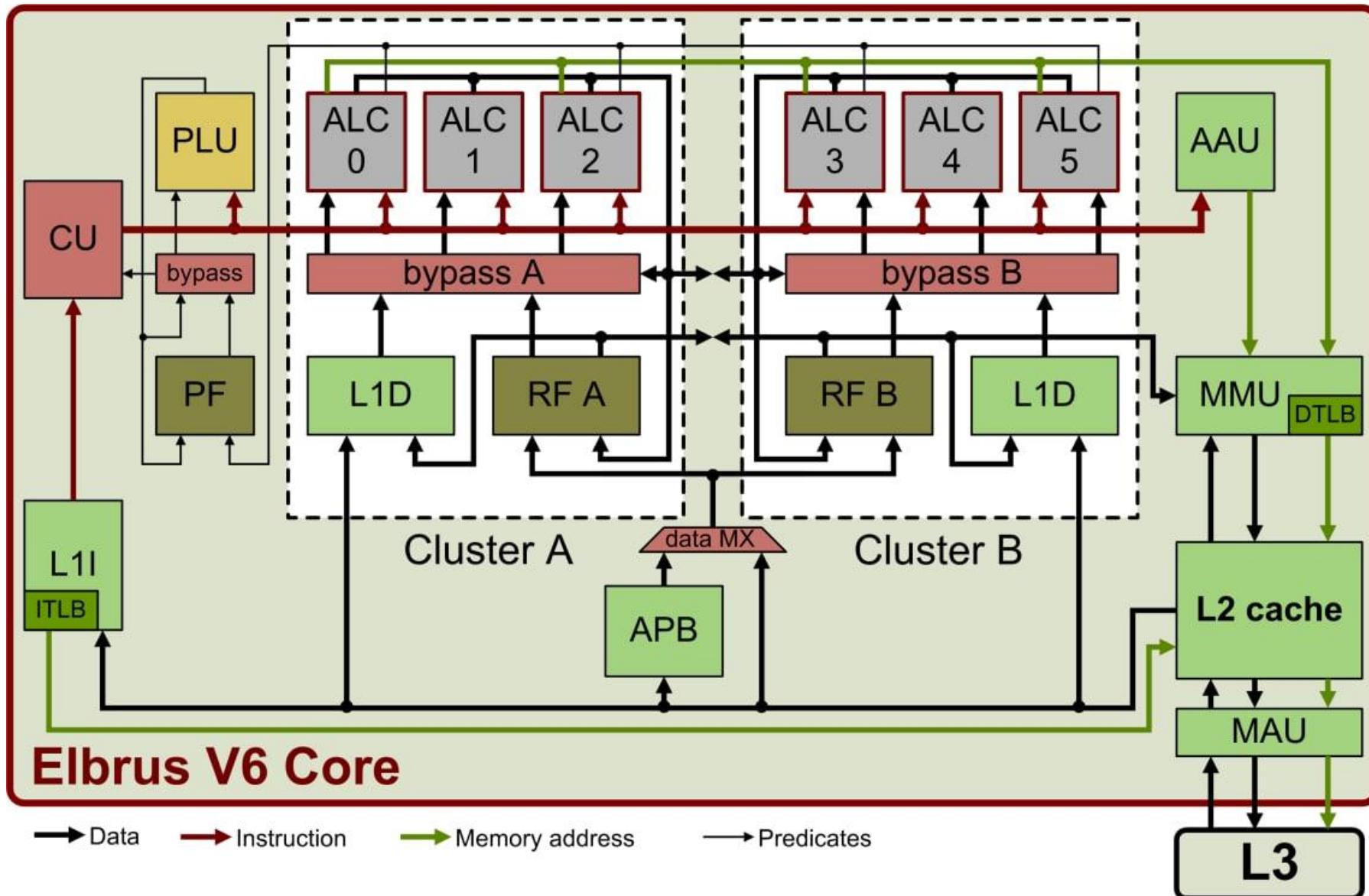
Архитектура Эльбрус. Широкая команда

Под широкой командой Эльбрус понимается набор элементарных операций Эльбрус, которые могут быть запущены на исполнение в одном такте.

В ШК «Эльбрус» доступны:

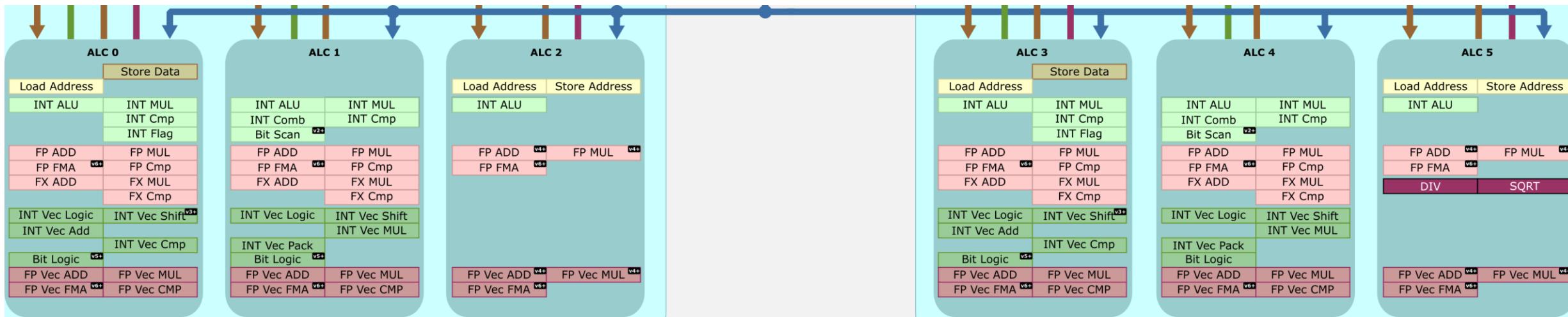
- 6 арифметико-логических устройств (АЛУ), исполняющих операции:
 - целочисленные (Int)
 - вещественные (FP)
 - сравнения (Cmp)
 - чтения из памяти (LD)
 - записи в память (ST)
 - операции над упакованными векторами (Vect)
 - деления и квадратного корня (Div/Sqrt)
- 1 устройство для операции передачи управления (CT);
- 3 устройства для операций над предикатами (PL);
- 6 квалифицирующих предикатов (QP);
- 4 устройства для команд асинхронного чтения данных по регулярным адресам в цикле (APB);
- 4 литерала размером 32 бита для хранения константных значений (LIT).

Архитектура Эльбрус. Структурная схема ядра.



MAU – Memory Access Unit
MMU – Memory Management Unit
AAU – Array Access Unit
CU – Control unit
IB – Instruction Buffer
ALC – Arithmetic Logic Channel
TLB – translation lookaside buffer
PLU – Predicate Logic Unit
RF – Register file
APB – Array Prefetch Buffer

Архитектура Эльбрус. АЛУ.



В процессорах Эльбрус (elbrus-v3+) имеется шесть АЛУ с номерами 0-5.

Основные поколения архитектуры Эльбрус: elbrus-v1 (устарела), elbrus-v2, elbrus-v3, elbrus-v4 (увеличено число операций FP), elbrus-v5 (добавлена поддержка SIMD-128), elbrus-v6 (поддержка FMA и аппаратной виртуализации).

Архитектура Эльбрус. АЛУ.

В составе АЛУ0 и АЛУ3 присутствуют:

- целочисленное арифметическое/побитовое/сдвиговое устройство;
- устройство сравнения;
- устройство для операций с упакованными целочисленными значениями шириной 8 бит, 16 бит, 32 бита;
- вещественное арифметическое устройство;
- вещественное арифметическое устройство над упакованными 32-разрядными вещественными числами;
- устройство обращения к памяти по чтению.

В составе АЛУ0 также присутствует устройство обращения к специальным регистрам.

В составе АЛУ1 и АЛУ4 присутствуют:

- целочисленное арифметическое/побитовое/сдвиговое устройство;
- устройство сравнения;
- устройство для операций с упакованными целочисленными значениями шириной 8 бит, 16 бит, 32 бита;
- вещественное арифметическое устройство;
- вещественное арифметическое устройство над упакованными 32-разрядными вещественными числами.

Архитектура Эльбрус. Матрица инструкций АЛУ.

Execution Unit	Command	Channels					
		0	1	2	3	4	5
Integer ALU, Logical, Shifts	two-operand	✓	✓	✓	✓	✓	✓
	three-operand		✓			✓	
	cmp	✓	✓		✓	✓	
Integer MUL	mul d/s	✓	✓		✓	✓	
Integer DIV	div						✓
Load/Store Address	ld		✓		✓	✓	
	st			✓			✓
Address transformation	Address transformation	✓	✓		✓	✓	
	Load/Store State Registers	✓					
Float-point ALU	add/sub d/s/pack	✓	✓	✓	✓	✓	✓
	add/sub ex	✓	✓		✓	✓	
	convert d/s/pack/ex	✓	✓		✓	✓	
	cmp d/s/ex	✓	✓		✓	✓	
Float-point MUL	d/s/pack	✓	✓	✓	✓	✓	✓
	ex	✓	✓		✓	✓	
Float-point three-operand	FP d/s/pack, fma	✓	✓	✓	✓	✓	✓
Float-point DIV, SQRT	fdiv s/d						✓
	remainder						✓
	sqrt s/d						✓
	fdiv ex						✓
Integer VEC	shift	✓	✓		✓	✓	
	logic	✓	✓		✓	✓	
	add/sub	✓			✓		
	mul		✓			✓	
Float-point VEC	add/sub	✓	✓	✓	✓	✓	✓
	mul	✓	✓	✓	✓	✓	✓
	cmp	✓	✓		✓	✓	
	FMA	✓	✓		✓	✓	

Вещественные устройства в АЛУ0, АЛУ1, АЛУ3 и АЛУ4, а также целочисленные устройства в АЛУ1 и АЛУ4 позволяют исполнять две последовательно зацепленные друг за друга операции в качестве одной трехаргументной операции, например, **shl_addd** ($a \ll 2 + 7$) или **fmul_sub** ($x * y + z$). Промежуточный результат первой операции не записывается в регистр, а используется только в качестве аргумента второй операции. Такие операции называют комбинированными.

В составе АЛУ2 и АЛУ5 присутствуют:

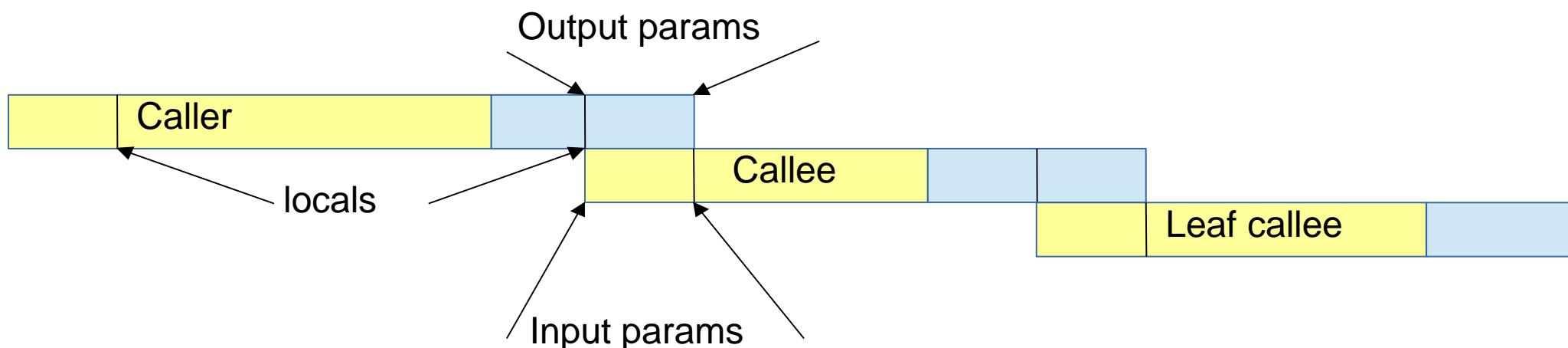
- целочисленное арифметическое/побитовое/сдвиговое устройство;
- устройство обращения к памяти по чтению/записи.

Также в АЛУ5 присутствуют устройство вещественного и целочисленного деления и устройство извлечения квадратного корня.

Архитектура Эльбрус. Регистровый файл.

Регистровый файл

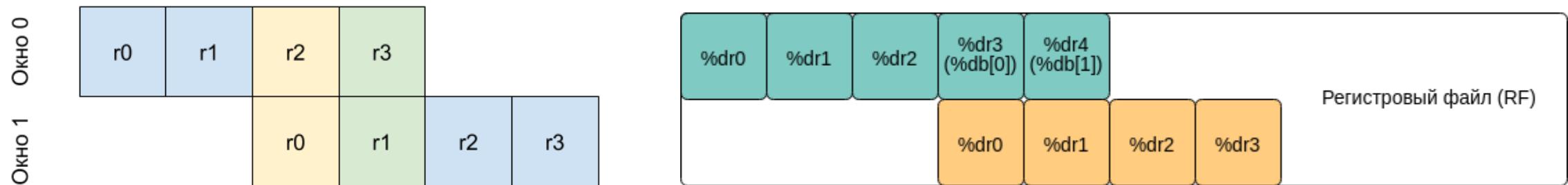
Параллельное исполнение операций по сравнению с последовательным требует необходимого количества оперативных регистров. Архитектура определяет регистровый файл объемом **256** 136-разрядных регистра (84-разрядные для старых версий Эльбрус) для целочисленных и вещественных данных, 32 регистра предназначены для глобальных данных и 224 регистра — для стека процедур.



Архитектура Эльбрус. Регистровое окно.

Назначения регистров:

- Регистры, в которых вызывающая функция передала нам параметры.
- Регистры, в которых мы просто работаем.
- Регистры, в которых передаются параметры вызываемой функции. В момент вызова функции регистровый файл виртуально “сдвигается” вниз - первая и вторая группы “уходят ниже ватерлинии” и не будут видны вызванной функции. Третья группа становится первой, а новые вторую и третью группы вызванная функция “заказывает” себе у процессора специальной командой.



Регистровым окном называется область (набор регистров) регистрационного файла, доступная в данный момент.

В коде имеется доступ к регистрам текущего окна — %dr0, %dr1, ..., %dr15

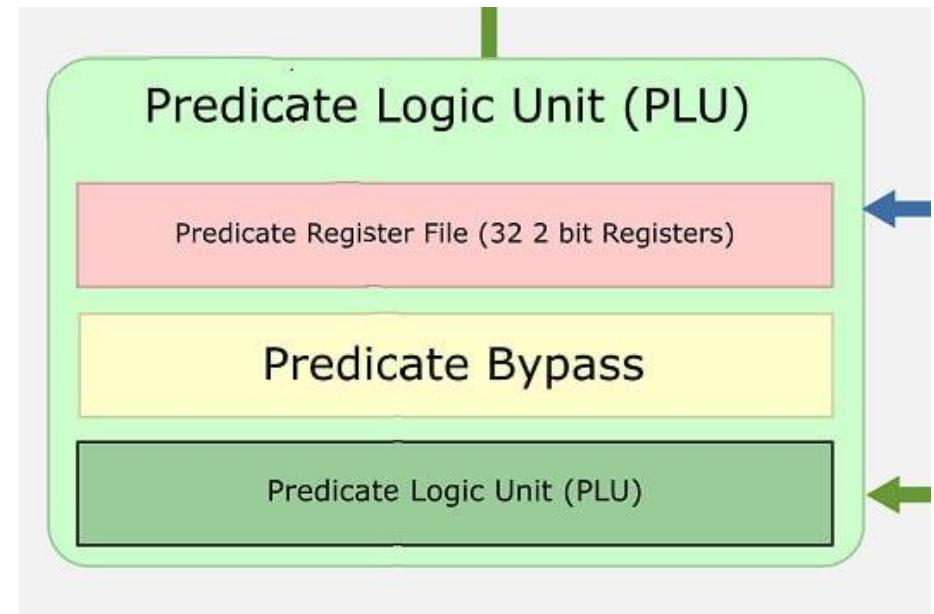
Размером регистрационного окна в Эльбрусе можно управлять.

Архитектура Эльбрус. Устройство логических предикатов (PLU)

Устройство логических предикатов (PLU)

Логический предикат (ЛП) — результат операции сравнения двух чисел (целых или вещественных), выполняемой в арифметико-логических каналах. Он представляется двухразрядным составным значением, содержащим тег (0 — обычный предикат, 1 — диагностический предикат) и булево значение (0 — false, 1 — true).

Устройство логических предикатов (УЛП) (Predicate Logic Unit (PLU)), разработанное с целью разгрузки арифметических устройств, выполняет различные операции над малоформатными значениями предикатов и выдает результаты в арифметико-логический канал для управления его действиями. В число операций АЛК входят считывание из предикатного файла, вычисление вторичных предикатов по двум первичным, задание предикатного (условного) выполнения операций в арифметико-логическом канале и др. Кроме этого, УЛП выполняет запись предикатов в предикатный файл (32 2-битных регистра).



Архитектура Эльбрус. Устройство логических предикатов (PLU)

В состав УЛП входят:

- блок вычисления логических предикатов (ВЛП) (PLU);
- блок распределения логических предикатов (РЛП) (Rout Logic Predicate Unit, RLPU);
- блок предикатного файла (ПФ) (Predicat File, PF) с байпасами.

Блок вычисления логических предикатов предназначен для получения вторичного предиката — результата вычисления булева выражения. Оно выполняется за три этапа:

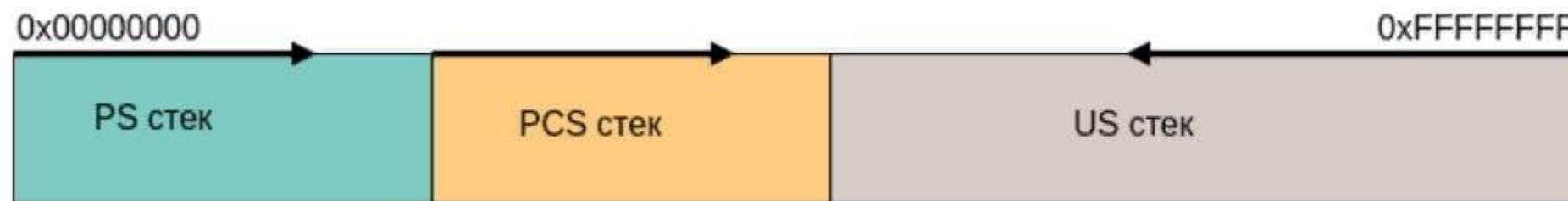
1. Считывание первичных operandов из предикатного файла отдельными операциями.
2. Логические операции над первичными operandами.
3. Запись результата в ПФ.

Блок распределения логических предикатов задает предикатный режим выполнения операции в арифметико-логическом канале (операция разрешается или запрещается) и передает ему предикат для управления.

Архитектура Эльбрус. 3 аппаратных стека

В архитектуре «Эльбрус» различают следующие разновидности стеков:

- стек процедур (Procedure Stack — PS);
- стек пользователя(User Stack — US);
- стек связующей информации (Procedure Chain Stack — PCS).



Архитектура Эльбрус. З аппаратных стека. Стек процедур

Стек процедур (Procedure Stack — PS). Стек процедур представляет собой непрерывную область, предназначенную для временного хранения тех фактических параметров и локальных данных еще не завершенных процедур, которые размещены компилятором в операционных регистрах (регистровых окнах). К этой категории данных относятся также результаты выполненных операций, записываемых в регистровый файл. Для каждой запускаемой процедуры в регистровом файле отводится окно (фрейм) требуемого размера, в пределах которого разрешена ее работа. Окно новой процедуры и предыдущее окно могут иметь общую область, в которой содержатся параметры, передаваемые запускающей процедурой, и возвращаемые значения — результаты ее работы. Завершение процедуры приводит к ликвидации ее окна. По мере запуска новых процедур ресурс свободных регистров РгФ может быть недостаточным для очередного окна. В этом случае нижняя часть стека автоматически откачивается в память. И наоборот, после завершения запущенной процедуры и возврата к прерванной процедуре необходимые данные автоматически подкачиваются из памяти в РгФ. В силу этого стек процедур состоит из двух частей, одна из которых располагается в регистровом файле, а для другой отводится участок памяти на случай переполнения РгФ.

Это могут быть аргументы функции, в “обычных” архитектурах это понятие ближе всего соответствуют регистрам общего назначения. В отличие от “обычных” процессорных архитектур, в Е2К регистры, используемые в функции, укладываются на отдельный стек.

Архитектура Эльбрус. Стек связующей информации.

Стек связующей информации (Procedure Chain Stack — PCS). Стек связующей информации предназначен для временного хранения данных о запускающей процедуре, обеспечивающих возврат к ней после отработки запускаемой процедуры. При защищенном программировании пользователь не должен иметь возможности изменять эту информацию, поэтому стек доступен только операционной системе и аппаратуре. Принцип организации стека связующей информации аналогичен стеку процедур.

Данные об адресе возврата, также как и в случае с регистрами, укладываются в отдельное место. Поэтому раскрутка стека (например, для выхода по исключению в C++) — более трудоемкий процесс чем в “обычных” архитектурах. С другой стороны, это исключает проблемы с переполнением стека.

Архитектура Эльбрус. Стек пользователя.

Стек пользователя (User Stack — US). Стек пользователя предназначен для данных, не относящихся к рассмотренной ранее категории, например различного рода динамических массивов или объектов программ на объектно-ориентированных языках.

В принципе вопросы выделения памяти под различные структуры данных решаются с помощью стандартных процедур управления динамической памятью типа new/delete. Но обращение к ним связано с дополнительными затратами времени при вызове процедур операционной системы. Поэтому в микропроцессоре реализован более эффективный способ управления динамической памятью, основанный на организации стека для этих целей. В систему команд введена специальная операция, формирующая дескриптор на участок памяти, выделенный из стека пользователя, вследствие чего процедура может эффективно заказывать себе память. Освобождение памяти выполняется автоматически при возврате из процедуры.

Архитектура Эльбрус. Программная конвейеризация циклов.

Позволяет наиболее эффективно исполнять циклы с независимыми (или слабо зависимыми) итерациями.

В программно-конвейеризированном цикле последовательные итерации выполняются с наложением - одна или несколько следующих итераций начинают выполняться раньше, чем заканчивается текущая. Шаг, с которым накладываются итерации, определяет общий темп их выполнения, и этот темп может быть существенно выше, чем при строго последовательном исполнении итераций. Такой способ организации исполнения цикла позволяет хорошо использовать ресурсы широкой команды и получать преимущество в производительности.

Архитектура микропроцессора содержит средства управления режимами выполнения пролога и эпилога цикла (разгонной и завершающей части конвейера), которые позволяют единым образом программировать выполнение всего цикла. В регистровом и предикатном файлах можно определять области для организации вращательного переименования регистров по принципу конвейерной ленты. Это позволяет запускать операцию со следующей итерации цикла до того, как был использован результат этой же операции на текущей итерации - конвейерная лента из регистров сохраняет значения в течение нескольких итераций.

Архитектура Эльбрус. Асинхронный доступ к массивам.

Позволяет независимо от исполнения команд основного потока буферизовать данные из памяти. Запросы к данным должны формироваться в цикле, а адреса линейно зависеть от номера итерации. Асинхронный доступ реализован в виде независимого дополнительного цикла, в котором кодируются только операции подкачки данных из памяти в FIFO-буфера. Из буфера данные забираются операциями основного цикла. Длина буфера и асинхронность независимого цикла позволяют устраниить блокировки по считыванию данных в основном потоке исполнения.

Устройство асинхронной подкачки массивов (APB)

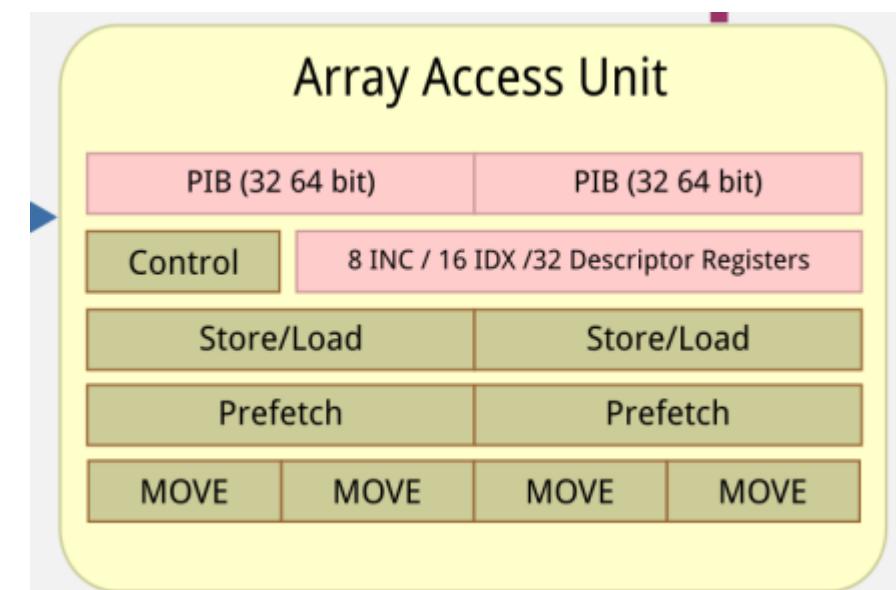
В одной широкой команде можно выполнить до 4 операций чтения из буфера APB.

Доступно регистров:

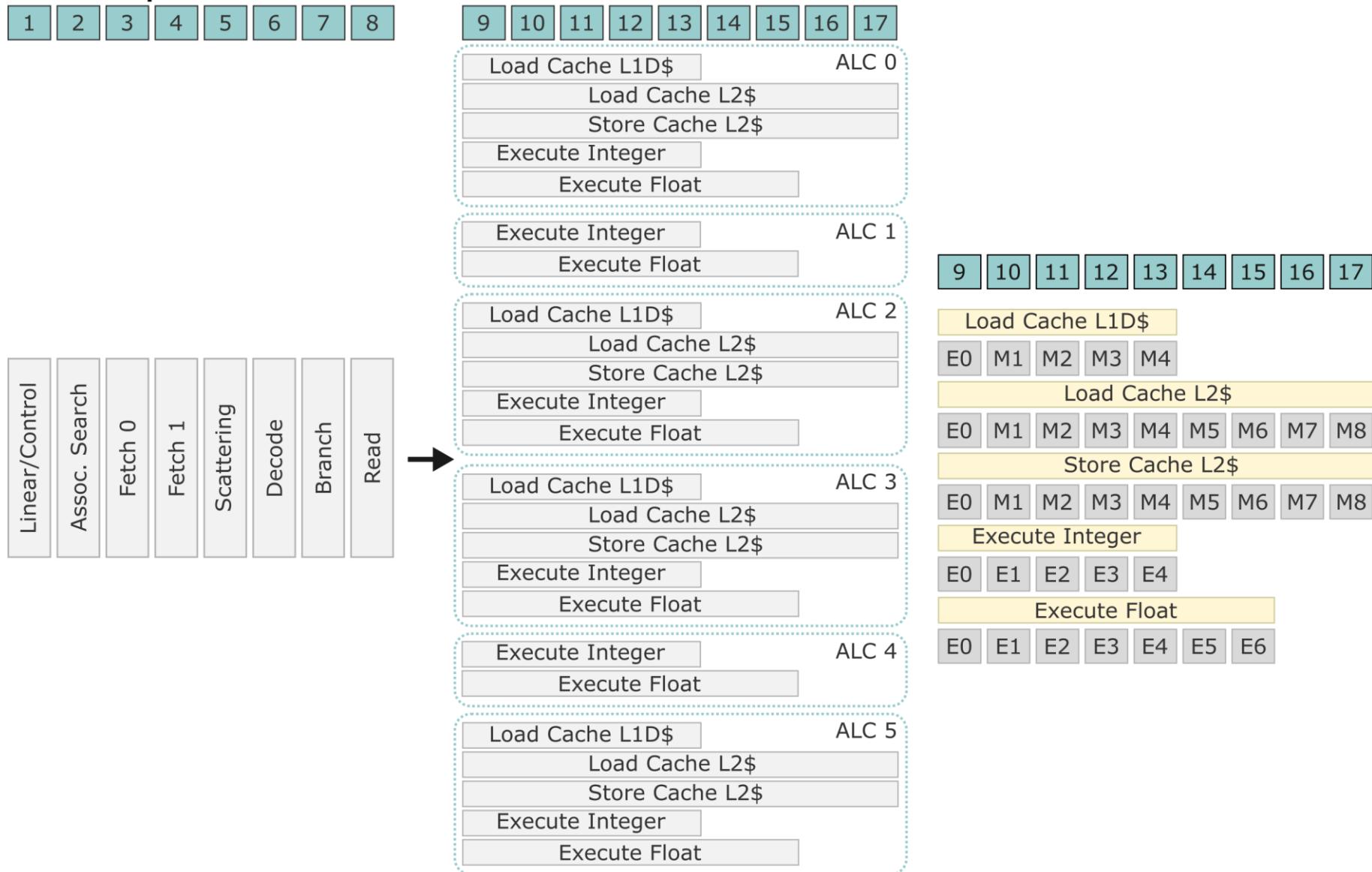
%aad – 32шт, формат дескриптора (здесь можно считать uint64)

%aaинд – 16шт, формат int32

%aaинк – 8шт, формат int32, %incr0 равен 1



Архитектура Эльбрус. Конвейер. E2K Pipeline



Архитектура Эльбрус. Команды.

ADDs/d	(.s)	sss/ddd	сложение целых 32/64 краткий комментарий	
			формат operandов и результата (результат занимает правую позицию): в примере операция ADDs принимает operandы одинарного формата и производит результат одинарного формата, тогда как операция ADDd - значения двойного формата; используются следующие правила:	
			s - operand или результат является значением одинарного формата в регистровом файле	
			d - operand или результат является значением двойного формата в регистровом файле	
			x - operand или результат является значением расширенного формата в регистровом файле	
			q - operand или результат является значением квадро формата в регистровом файле	
			b - operand или результат является предикатом в предикатном файле	
			v - operand или результат является предикатом, вычисленным в текущей широкой команде	
			e - результат операции управляет выполнением других операций в текущей широкой команде (предикатное выполнение)	
			r - operand или результат является регистром состояния	
			i - operand является непосредственной константой из текущей широкой команды	
			- - отсутствие операнда	
			признак спекулятивного исполнения операции	
мнемоника операции (в примере - ADDs или ADDd)				

Команды процессора Эльбрус

```
{  
    nop 4  
    muls,0,sm    %g24, %r2, %b[13]  
    muls,1,sm    %g17, %r2, %b[14]  
    muls,3,sm    %g25, %r2, %b[11]  
    muls,4,sm    %g18, %r2, %b[12]  
  
    }  
    {  
        loop_mode  
        muls,0,sm    %b[7], %r2, %b[9]  
        muls,1,sm    %b[8], %r2, %b[10]  
        ldw,2,sm     %dr4, %db[21], %b[1]  
        ldw,3,sm     %dr6, %db[19], %b[0]  
        adds,4,sm    %r0, %b[15], %b[20]  
        adds,5,sm    %r0, %b[16], %b[18]  
    }  
}
```

Архитектура процессоров Эльбрус.

Ассемблер.

```
void a(int n, int a, int b, int c, int *x) {  
    for(int i = 0; i < n; i++) {  
        x[i] = (x[i] + a) * b + c;  
    }  
}
```

```

a(int, int, int, int, int*):
{
    setwd wsz = 0x6, nfx = 0x1, dbl = 0x1
    return %ctpr3
}
{
    cmplsb,0 0x0, %r0, %pred0
}
{
    merges,0,sm 0x1, %r0, %r9, %pred0
    addd,1 0x0, 0x0, %dr10 ? %pred0
}
{
    shrs,0,sm  %r9, 0x1, %r8
}
{
    ct %ctpr3 ? ~%pred0
    cmpesb,0,sm %r8, 0x0, %pred1
    cmplsb,1,sm 0x0, %r8, %pred2
}
{
    nop 2
    return %ctpr3
    muls,0,sm %r1, %r2, %r7
    merges,1,sm 0x1, %r8, %r0, %pred2
    pass %pred0, @p0
    pass %pred1, @p1
    landp @p0, @p1, @p4
    pass @p4, %pred0
}
{
    ibranch .L416 ? %pred0
}
{
    setwd wsz = 0x11, nfx = 0x1, dbl = 0x1
    setbn rsz = 0xa, rbs = 0x6, rcur = 0x0
    disp %ctpr1, .L20
    addd,0 %dr4, 0x4, %dr6
    addd,1 0x0, _f64,_lts1 0x40ff2000000000, %dg16
    addd,2 0x2, 0x0, %dg17
    adds,3 0x0, 0x0, %r11
}
{
    return %ctpr3
    addd,0,sm 0x0, 0x0, %db[11]
    insfd,1 %dg16, _f32s,_lts0 0x8800, %dr0, %dg16
    aaurwd,2 %dr10, %aasti1
    aaurwd,5 %dr4, %aad0
}
{
    addd,0,sm %db[11], _f16s,_lts0lo 0x10, %dg18
    addd,1,sm 0x8, %db[11], %dg17
    aaurwd,2 %dg17, %aaincr1
}
{
    }
    {
        ldw,0,sm %dr4, %db[11], %g19
        addd,1,sm %db[11], _f16s,_lts0lo 0x18, %dg21
        ldw,2,sm %dr6, %db[11], %g20
        addd,3,sm %db[11], _f16s,_lts0hi 0x20, %dg22
        addd,4,sm %db[11], _f16s,_lts1lo 0x28, %dg23
        addd,5,sm %db[11], _f16s,_lts1hi 0x30, %db[21]
    }
    {
        ldw,0,sm %dr6, %dg18, %g18
        addd,1 %dg16, _f64,_lts0 0xffffffff, %dg26
        ldw,2,sm %dr4, %dg17, %g24
        ldw,3,sm %dr6, %dg17, %g17
        addd,4,sm 0x8, %db[21], %db[19]
        ldw,5,sm %dr4, %dg18, %g25
    }
    {
        rwd,0 %dg16, %lsr
        shld,1 %dg26, 0x1, %dr10
        ldw,2,sm %dr4, %dg21, %b[7]
        ldw,3,sm %dr6, %dg21, %b[8]
        ldw,5,sm %dr4, %dg22, %b[5]
    }
    {
        rwd,0 %dr0, %lsr1
        adds,1 %r3, %r7, %r0
        ldw,2,sm %dr6, %dg22, %b[6]
        ldw,3,sm %dr4, %dg23, %b[3]
        ldw,5,sm %dr6, %dg23, %b[4]
    }
    {
        ldw,0,sm %dr6, %db[21], %b[2]
    }
    {
        muls,0,sm %g19, %r2, %b[15]
        muls,1,sm %g20, %r2, %b[16]
    }
    {
        nop 4
        muls,0,sm %g24, %r2, %b[13]
        muls,1,sm %g17, %r2, %b[14]
        muls,3,sm %g25, %r2, %b[11]
        muls,4,sm %g18, %r2, %b[12]
    }
    {
        loop_mode
        muls,0,sm %b[7], %r2, %b[9]
        muls,1,sm %b[8], %r2, %b[10]
        ldw,2,sm %dr4, %db[21], %b[1]
        ldw,3,sm %dr6, %db[19], %b[0]
        adds,4,sm %r0, %b[15], %b[20]
    }
}
{
    adds,5,sm %r0, %b[16], %b[18]
}
{
    loop_mode
    alc alcfc=1, alct=1
    abn abnf=1, abnt=1
    ct %ctpr1 ? %NOT_LOOP_END
    staaw,2 %b[20], %aad0[ %aasti1 ]
    addd,4,sm 0x8, %db[19], %db[17]
    staaw,5 %b[18], %aad0[ %aasti1 + _f32s,_lts0 0x4 ]
    incr,5 %aaincr1
}
{
    setwd wsz = 0x6, nfx = 0x1, dbl = 0x1
    shld,0,sm %dr10, 0x2, %dr0
    shls,1 %r8, 0x1, %r6
}
{
    aaurw,2 %r11, %aabf0
}
{
    nop 2
    ldw,0,sm %dr4, %dr0, %g16
    cmpesb,1 %r6, %r9, %pred0
}
{
    ct %ctpr3 ? %pred0
}
nop
{
    adds,0,sm %g16, %r1, %g16
}
{
    nop 5
    muls,0,sm %g16, %r2, %g16
}
{
    adds,0,sm %g16, %r3, %g16
}
{
    ct %ctpr3 ? ~%pred0
    stw,2 %dr4, %dr0, %g16 ? ~%pred0
}
{
    ibranch .L82
    shld,0,sm %dr10, 0x2, %dr0
    shls,1 %r8, 0x1, %r6
}
elbrus_optimizing_compiler_v1.28.09_Nov_28_2023 = 0x0

```

Архитектура Эльбрус. Ассемблер. Передача аргументов.

```
foo(int, int, int, int, int, int, int):
{
    nop 3
    setwd wsz = 0x9, nfx = 0x1, dbl = 0x0
    setbn rsz = 0x3, rbs = 0x5, rcur = 0x0
    disp %ctpr1, _Z3bariiiiiii
    getsp,0      _f32s,_lts1 0xffffffffc0, %dr9
}
{
    sxt,0 0x2, %r1, %db[1]
    sxt,1 0x2, %r0, %db[0]
    sxt,2 0x2, %r7, %db[7]
    sxt,3 0x2, %r6, %db[6]
    sxt,4 0x2, %r5, %db[5]
    sxt,5 0x2, %r4, %db[4]
}
{
    sxt,0 0x2, %r3, %db[3]
    sxt,1 0x2, %r2, %db[2]
}
{
    call %ctpr1, wbs = 0x5
```

```
    {
        return %ctpr3
        adds,3 %b[0], 0x1, %r1
    }
    {
        nop 4
        sxt,3 0x2, %r1, %dr0
    }
    {
        ct %ctpr3
    }
```

elbrus_optimizing_compiler_v1.28.09_Nov_28_2023

```
int bar(int, int, int, int, int, int, int, int);
int foo(int a, int b, int c, int d, int e, int f, int g, int h) {
    return bar(a, b, c, d, e, f, g, h) + 1;
}
```

<https://ce.mentality.rip>



Архитектура Эльбрус. Эмулятор Qemu-e2k.

```
$ git clone --depth=1 -b e2k https://git.mentality.rip/OpenE2K/qemu-e2k.git
$ cd qemu-e2k
$ mkdir build
$ cd build
$ ../configure --target-list=e2k-linux-user --static --disable-capstone --disable-werror
$ nice ninja
$ sudo cp qemu-e2k /usr/local/bin

$ cat hello.c
#include <stdio.h>
int main(int argc, char *argv[]) {
    const char *name = argc > 1 ? argv[1] : "world";
    printf("Hello, %s!\n", name);
    return 0;
}
$ lcc -O2 -static hello.c -o hello-e2k
$ qemu-e2k hello-e2k $USER
Hello, vasya!
```



<https://git.mentality.rip/OpenE2K/qemu-e2k.git>

Программа Начального Старта (ПНС)

ПНС. Стартовое меню.

- 'c' - Change boot parameters
- 'u' - Show cUrrent parameters
- 'd' - Show Disks and partitions
- 'm' - Save params to NVRAM
- 'b' - Start Boot.conf menu
- 'z' - Change configuration LM63
- 'w' - Watch Brief Boot Balance about Hardware-Config
- 'T' - Tinyspi Led-Blink Test
- 'W' - Set OS WatchDog Reset in On/Off
- 't' - Enter debug tests mode
- '.' , ',' - Enter enhanced cmd mode
- 'K' - Set PHY-Drift Mode in On/Off
- 'N' - No NVRAM Clearing for Next One Start
- 'G' - Set GUI Run in On/Off
- 'M' - Set More Mode in On/Off

BOOT SETUP

Press command letter, or press 'h' to get help

ПНС. Меню загрузки.

Необходимо нажать пробел, после чего должны появиться следующие строки:

Key pressed. Autoboot canceled.

CPU#00: Starting menu.

BOOT SETUP

Press command letter, or press 'h' to get help

:

- d — show Disks and partitions (показать диски и разделы);
- c — Change boot parameters (изменить параметры загрузки);
- u — show cUrrent parameters (показать текущие параметры);
- m — save params to NVRAM (сохранить параметры в NVRAM);
- b — start Boot.conf menu (запустить меню Boot.conf).

ПНС. СПИСОК ДИСКОВ.

При нажатии на клавишу **d** получим список дисков:

```
:d
CPU#00: Drive [2]: SATA - PCI
BUS[1]:DEV[3]:FUNC[0], MCST SATA COMBINED Port
[0] - KINGSTON SMS200S3120G
CPU#00:          Partition [0]: Linux EXT2;
                  U:246194e7-0512-4db3-a821-
cbcbe3c92c38 L:""
CPU#00:          Partition [1]: Linux swap
CPU#00:          Partition [3]: Extended
CPU#00:          Partition [4]: Unknown file
system type
CPU#00: Drive [10]: ATAPI device
```

ПНС. Параметры загрузки.

```
:c                                         Enter command string : < Skipped >
                                             Enter filename       : < Skipped >
                                             Enter initrd file name: < Skipped >
                                             Enter autoboot value  : < Skipped >

          CHANGE BOOT PARAMETERS

    Current Settings:
drive_number:      '2'                      Current Settings:
drive_label:        '*'                      drive_number:      '10'
partition_number:  '0'                      drive_label:
file system id:   '07bde958-ec62-492e-        partition_number:  '0'
933c-17334bb02da2'                           file system id:
command_string:                                command_string:
filename:                                     filename:
initrdfsfilename:                            initrdfsfilename:
autoboot in:        '10'                     autoboot in:      '10'

CPU#00: Search drive and partition by label
To advance to next setting press ENTER. To or uuid succeed
skip setting press ESC
Enter drive number   : 10
Enter partition number: < Skipped >
```

Средства разработки под процессоры

Эльбрус

Компилятор LCC

Фирменный компилятор компании АО «МЦСТ» — разработчика архитектуры Эльбрус. Поддерживает языки программирования **C**, **C++**, **Fortran**. Во многом совместим с компилятором GCC (GNU Compiler Collection) — как по параметрам запуска, так и по GNU-расширениям языков. Обладает развитыми средствами оптимизации генерируемого машинного кода, позволяющими выбирать между быстродействием программы и её размером, а также длительностью компиляции.

Компилятор предоставляет богатые возможности оптимизации под платформу Эльбрус, причём от ветви к ветке производительность одного и того же исходного кода на одной и той же аппаратуре в среднем растёт; обратите также внимание на библиотеку **EML**.

Компилятор LCC написан на языке **C**, актуальная рабочая версия, и представляет из себя более 2,76 млн. строк кода, 0,74 млн. строк кода из которых принадлежат начальной фазе компиляции (frontend) от фирмы EDG. Совместимость с компилятором gcc на уровне опций и расширений позволяет собирать полноценную ОС Эльбрус, основанную на ОС GNU/Linux, включая сборку непосредственно ядра Linux.

[Обзор компилятора lcc для микропроцессора Эльбрус. А.Л. Маркин, АО «МЦСТ»](#)

Компилятор LCC. Версии.

Версия LCC	1.28	1.27	1.26	1.25	1.24	1.23	1.21	1.19	
Год выпуска	2023	2022	2021	2020	2019	2018	2016	2014	
Поддержка C++	GCC ≈	11.3	9.3	9.3	7.3	7.3	5.5	4.8	4.4
	libstdc++	6.0.29	6.0.28	6.0.28	6.0.24	6.0.24	6.0.21	6.0.18	6.0.11
	C++23	±	±	±	±	-	-	-	-
	C++20	±	±	±	±	-	-	-	-
	C++17	+	±	±	±	±	-	-	-
	C++14	+	+	+	+	+	+	± ²	-
	C++11	+	+	+	+	+	+	± ²	-
	C++03	+	+	+	+	+	+	+	+
	C++98	+	+	+	+	+	+	+	+
Поддержка C	GCC ≈	11.3	9.3	9.3	7.3	7.3	5.5	4.8	4.4
	libc	2.35	2.35	2.29	2.29	2.29	2.23	2.23	2.19
	C18	+	+	+	-	-	-	-	-
	C11	+	+	+	+	+ ¹	+ ¹	+ ¹	-
	C99	+	+	+	+	+	+	+	+
	C90	+	+	+	+	+	+	+	+

Компилятор LCC. Средства.

Компонент	Назначение \ Номер версии	29	28	27	26	25	24	23	21	19	16
binutils	средства работы с машинным кодом	2.41	2.41	2.39	2.36	2.35	2.34	2.29	2.26	2.23	2.18
dprof	профилировщик машинного кода	1.4.0	1.4.0	1.4.0	1.4.0	1.3.8	1.3.8	1.3.8	1.3.4	1.3.2	н/д
gcov	анализатор покрытия машинного кода	4.7.3	4.7.3	4.7.3	4.2.1	4.2.1	4.2.1	4.2.1	4.2.1	4.2.1	н/д
gdb	отладчик программ в машинных кодах	13.2	13.2	9.1	9.1	9.1	8.3.1	8.1	7.11	7.2	5.2
lcc	компилятор C, C++, Fortran и дизассемблер	1.29	1.28	1.27	1.26	1.25	1.24	1.23	1.21	1.19	1.16

Компилятор LCC. Директивы C/C++.

Для платформы Эльбрус предусмотрен макрос языка С:

`__e2k__`

Компилятор LCC вводит свой макрос (для архитектур Эльбрус и Спарк):

`__LCC__`

Примеры использования:

```
#if (defined __LCC__) && (! defined __OPTIMIZE__)
    #define MY_MACROS MY_MACROS_LCC
#endif

#ifndef __e2k__
    #include "implementation_elbrus.h"
#endif
```

Компилятор LCC. Уровни оптимизации

-00

Начальный уровень. Сохраняет соответствие между исходным текстом программы и ее двоичным кодом, что позволяет формировать отладочную информацию для символьного отладчика.

-01

Локальные оптимизации потока данных и управления, не требующие аппаратной поддержки, выполняются в рамках линейного участка.

-02

Внутрипроцедурные оптимизации потока данных и управления, использующие все аппаратные возможности, цикловые оптимизации, слабо увеличивающие размер кода, минимальные межпроцедурные оптимизации.

-03

Все межпроцедурные оптимизации, возможность оптимизации в режиме «вся программа», все цикловые оптимизации.

-04

Включает все предыдущие и дополнительные агрессивные оптимизации. Стоит пробовать экспериментально. Может приводить как к повышению производительности, так в некоторых случаях и к деградации производительности.

Для генерации кода с привязкой к исходному тексту нужно использовать опции -O0 -g.

[Руководство по эффективному программированию на платформе «Эльбрус»](#)

Компилятор LCC. Опции профилирования.

Опции данной секции реализуют технику двухфазной компиляции:

на первой фазе программа собирается в инструментирующем режиме. Затем программа выполняется на тестовых данных или в выбранном сценарии использования, который требует ускорения. Программа выполняется по сценариям (одному или нескольким), в результате чего в файле формируется профиль исполнения. Данный профиль затем используется для второй фазы компиляции с помощью **-fprofile-use**.

Основным требованием для применения данного режима является наличие набора представительных сценариев использования программы. Если в дальнейшем реальное исполнение программы существенно отличается от того варианта, на котором получали профиль, то производительность реального исполнения может значительно ухудшиться.

-fprofile-generate[=<path>]

генерировать компиляторный профиль;

-fprofile-use[=<file>]

использовать компиляторный профиль.

Компилятор LCC. Другие режимы компиляции.

Режим «вся программа» (**fwhole**) позволяет выполнять межпроцедурные оптимизации. В этом режиме анализируемый контекст не ограничивается единственной функцией.

Опции необходимо подавать как при генерации объектных файлов .o, так и при линковке итогового файла (исполнимого либо динамической библиотеки).

Основным требованием для данных опций является необходимость разделять цели в сборочной системе программы, и для разных целей по-своему модифицировать флаги сборки:

- для динамических библиотек использовать **-fPIC -fwhole-shared**;
- для исполняемых файлов использовать **-fwhole** либо **-fPIE -fwhole-shared**;

Если собираемый пакет содержит одновременно исполняемые файлы и библиотеки, то глобально на уровне CFLAGS возможно выставить только сочетание **-fPIC -fwhole-shared**.

-fwhole

опция компиляции в режиме «вся программа». Несовместима с позиционно-независимым кодом, в частности, с динамическими библиотеками .so.

-fwhole-shared

опция компиляции в режиме «вся программа», совместимая с позиционно-независимым кодом. Требует работы с **-fPIC** либо **-fPIE**.

Компилятор LCC. OpenMP.

Возможности:

- Поддержан стандарт OpenMP 3.1.
- Доступны языки C, C++, Fortran.

Ограничения:

- Для e2k не поддержано в режиме -m128.
- Nested параллелизм не поддержан. Если при исполнении уже распараллеленного цикла встречаются циклы, которые нужно распараллелить, то эти (вложенные) циклы будут исполняться последовательно.
- Не поддержан clause collapse.
- Для C/C++ после директивы **#pragma omp** всегда должен следовать statement языка. Проблемы могут возникнуть для **#pragma omp barrier** и **#pragma omp flush**, если за ними нет statement'a. Для обхода проблемы рекомендуется в следующей строке поставить пустой statement, например "0;" или ";"
- Переменные, перечисленные в clause'ах private, lastprivate, firstprivate и threadprivate должны иметь скалярный базовый тип или массив скалярного базового типа. В противном случае результат программы неопределен.
- Директива **#pragma omp for** не поддержана для итераторов C++.
- Для C/C++ clause'ы **if** и **num_threads** своими параметрами могут иметь только константы и переменные целого типа, выражения не допускаются.

Компилятор LCC. Средства для расчётов.

- EML (Elbrus Math Library: библиотека оптимизированных алгоритмов)
- Builtin E2K asm, например:
 - `__builtin_e2k_qpfadd`, `__builtin_e2k_qpfmul`, `__builtin_e2k_qpfsub`
- Intel интринсики (на уровне интерфейса функций, реализация низкоуровневая)
- Реализации MPI: **mpich** и **openmpi**
- Интерфейсы LAPACK, BLAS
- gcc builtins
 - Ряд билтинов (builtin) GCC поддержан с ограничениями.
 - Некоторые из них вызваны особенностями реализации компилятора, другие отсутствием практической необходимости. С выходом новых версий lcc расширяется состав поддержанных билтинов.

Стандартные средства разработки

Средства разработки. НРС.

Версия ОС	8.x	7.x	6.x	5.0	4.1	4.0	3.1	3.0
BLAS	(EML)	(EML)	(EML)	(EML)	(EML)	(EML)	(EML)	(EML)
CLC	13.0.1	13.0.1	20190827	20190827	0.2.0	0.2.0	-	-
EML	11.0	10.0	9.3	8.0	7.0	7.0	6.0	6.0
FFTW	3.3.8 2.1.5	3.3.8 2.1.5	2.1.5	2.1.5	2.1.5	2.1.5	2.1.5	2.1.5
LAPACK	(EML)	(EML)	(EML)	(EML)	(EML)	(EML)	(EML)	(EML)
MPICH	4.1.2	4.1.2	3.1.4 1.1.1p1	3.1.4 1.1.1p1	3.1.4 1.1.1p1	3.1.4 1.1.1p1	3.1.4 1.1.1p1	3.1.4 1.1.1p1
OpenCV	4.5.5	4.5.5	3.2.0	3.2.0	3.2.0	3.2.0	3.2.0	3.2.0
OpenMPI	4.1.5	4.1.5	4.0.5	2.0.2	2.0.2	2.0.2	2.0.2	2.0.2
ScaLAPACK	2.1.0	2.1.0	2.1.0	-	-	-	-	-
SLURM	20.11.4	20.11.4	20.11.4	19.05.5	17.02.11	17.02.11	17.02.11	17.02.11

Средства разработки. Версии ЯП.

```
$ python -V
Python 2.7.15

$ python3 -V
Python 3.9.10

$ lua -v
Lua 5.1.5 Copyright (C) 1994-2012 Lua.org, PUC-Rio

$ java -version
openjdk version "1.8.0_252"

$ php -v
PHP 7.4.7 (cli) (built: Aug 20 2022 07:22:31) ( NTS )

$ ruby -v
ruby 2.7.3p183 (2021-04-05 revision 6847ee089d) [x86_64-linux]

$ perl -v
This is perl 5, version 30, subversion 3 (v5.30.3) built for x86_64-linux

$ /opt/mcst/gcc-9.0.0-A.XXX.e2k-v5.5.10/bin/go version
go version unknown linux/amd64

$ mono -V
Mono JIT compiler version 5.16.0.220 (5.16.0.220/e2k_3.2 Fri Aug 19 16:50:07 MSK 2022)

$ node -v
v12.16.3
```

Средства разработки. ЯП.

Современные языки программирования:

- JVM (Java/Kotlin/Scala/Groovy) – JIT компилятор, JVM 8 и 11, хорошая поддержка JNI + JavaCPP + JNA
- C/C++ - полная совместимость LCC и GCC, поддержка C++11, C++14, C++17 (удалось собрать и запустить БД на данном диалекте). Поддержка корутин, портированный набор системных вызовов Linux Kernel.
- Lua – полная поддержка языка, поддержка LuaJIT, достаточно легкий процесс портирования и запуска модулей
- Ruby, Python, PHP, JS – полная поддержка языков и основных библиотек; есть вопросы к быстродействию и поддержки native расширений
- Go, Rust – экспериментальный статус. Программы собираются и запускаются, однако есть много вопросов к быстродействию, а также к экосистеме библиотек

Средства разработки. Версии ЯП.

```
llvm13: 13.0.1-vd7u91
clang13: 13.0.1-vd7u91
liblccopt: 141516-vd7u265
lccrt: 141516-vd7u265
rust: 1.57.0_elbrus-v4_27062023
Gccgo: 1.17
```

```
numas13@yukari ~/dev/rust/joshuto/src
drwxr-xr-x 1/15 2023-09-06 15:44 UTC 4.00 K
    con...          commands      32
    docs           config        7
    src            context       8
    ...lock
    ...toml
    Doc...
    LIC...
    ...md
    ...toml
    ...png
    ui
    util          key_command  12
                  preview       4
                  ui             6
                  util          11
    @ history.rs      6.05 K
    @ main.rs        4.60 K
    @ run.rs         4.92 K
    @ tab.rs         2.56 K
    @ bulk_rename.rs
    @ change_directory.rs
    @ command_line.rs
    @ cursor_move.rs
    @ delete_files.rs
    @ file_ops.rs
    @ flat.rs
    @ line_nums.rs
    @ mod.rs
    @ new_directory.rs
    @ numbered_command.rs
    @ open_file.rs
    @ parent_cursor_move.rs
    @ preview_cursor_move.rs
    @ quit.rs
    @ reload.rs
    @ rename_file.rs
    @ search.rs
    @ search_fzf.rs
    @ search_glob.rs
```

Средства разработки. Dotnet core (C#), Java.

```
$ /opt/mcst/dotnet/dotnet --info  
Пакет SDK для .NET (отражающий любой global.json):  
Version: 6.0.10.3  
Commit: 1231ff0e6f
```

```
Среда выполнения:  
OS Name: elbrus  
OS Version: 7  
OS Platform: Linux  
RID: elbrus-e2k  
Base Path: /opt/mcst/dotnet/dotnet/sdk/6.0.110/
```

```
global.json file:  
Not found
```

```
Host:  
Version: 6.0.10  
Architecture: e2k  
Commit: N/A
```

```
.NET SDKs installed:  
6.0.110 [/opt/mcst/dotnet/dotnet/sdk]
```

```
.NET runtimes installed:  
Microsoft.AspNetCore.App 6.0.10 [/opt/mcst/dotnet/dotnet/shared/Microsoft.AspNetCore.App]  
Microsoft.NETCore.App 6.0.10 [/opt/mcst/dotnet/dotnet/shared/Microsoft.NETCore.App]
```

```
Download .NET:  
https://aka.ms/dotnet-download
```

```
Learn about .NET Runtimes and SDKs:  
https://aka.ms/dotnet/runtimes-sdk-info
```

```
$ java -version  
openjdk version "1.8.0_252"  
OpenJDK Runtime Environment (build 1.8.0_252-b09)  
OpenJDK 64-Bit Server VM (RVM 3.7.0 by Unipro) (build 25.252-b09,  
mixed mode)
```

Средства разработки. IDE Code:Blocks.

The screenshot shows the Code:Blocks IDE interface. The main window displays the file `whets.c` with the following content:

```
196 #endif
197
198 void whetstones(long xtra, long x100, int calibrate);
199 void pa(SPDP el[4], SPDP t, SPDP t2);
200 void po(SPDP el[4], long j, long k, long l);
201 void p3(SPDP *x, SPDP *y, SPDP *z, SPDP t1, SPDP t2);
202 void pout(char title[22], float ops, int type, SPDP checksum,
203           SPDP time, int calibrate, int section);
204
205 static SPDP loop_time[9];
206 static SPDP loop_mops[9];
207 static SPDP loop_mflops[9];
208 static SPDP TimeUsed;
209 static SPDP mwips;
210 static char headings[9][18];
211 static SPDP Check;
212 static SPDP results[9];
213
214
215 int main(int argc, char *argv[])
216 {
217     int count = 10, calibrate = 1;
218     long xtra = 1;
219     int section;
220     long x100 = 100;
221     int duration = 10;
222     FILE *outfile;
223     char compiler[80], options[256], general[10][80] = {" "};
224     char endit[80];
225     int i;
226     int nopause = 1;
227
228     if (argc > 1)
229     {
230         switch (argv[1][0])
231     }
```

The left sidebar shows the project management and symbol browser. The symbols list includes:

- Check : float
- headings : char
- loop_mflops : float
- loop_mops : float
- loop_time : float
- mwips : float
- results : float
- TimeUsed : float

The bottom window shows the build logs:

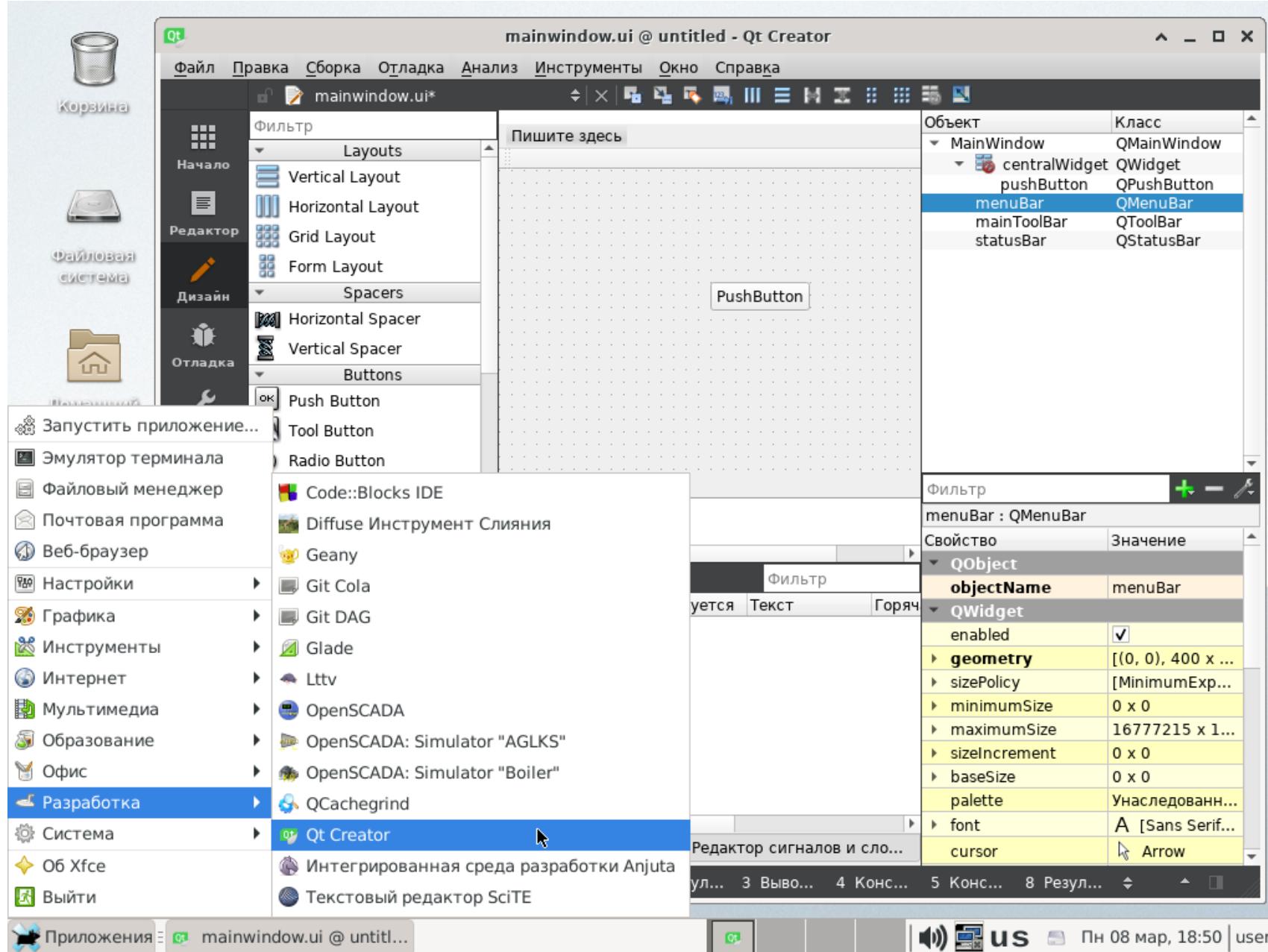
```
----- Build file: "no target" in "no project" (compiler: unknown) -----
gcc -c /export/home/entityfx/anybench/sources/whets.c -o /export/home/entityfx/anybench/sources/whets.o
lcc: "/export/home/entityfx/anybench/sources/whets.c", строка 242: предупреждение:
    функция "getDetails" объявлена неявно
    [-Wimplicit-function-declaration]
    getDetails();

lcc: "/export/home/entityfx/anybench/sources/whets.c", строка 244: предупреждение:
    функция "local_time" объявлена неявно
    [-Wimplicit-function-declaration]
    local_time();

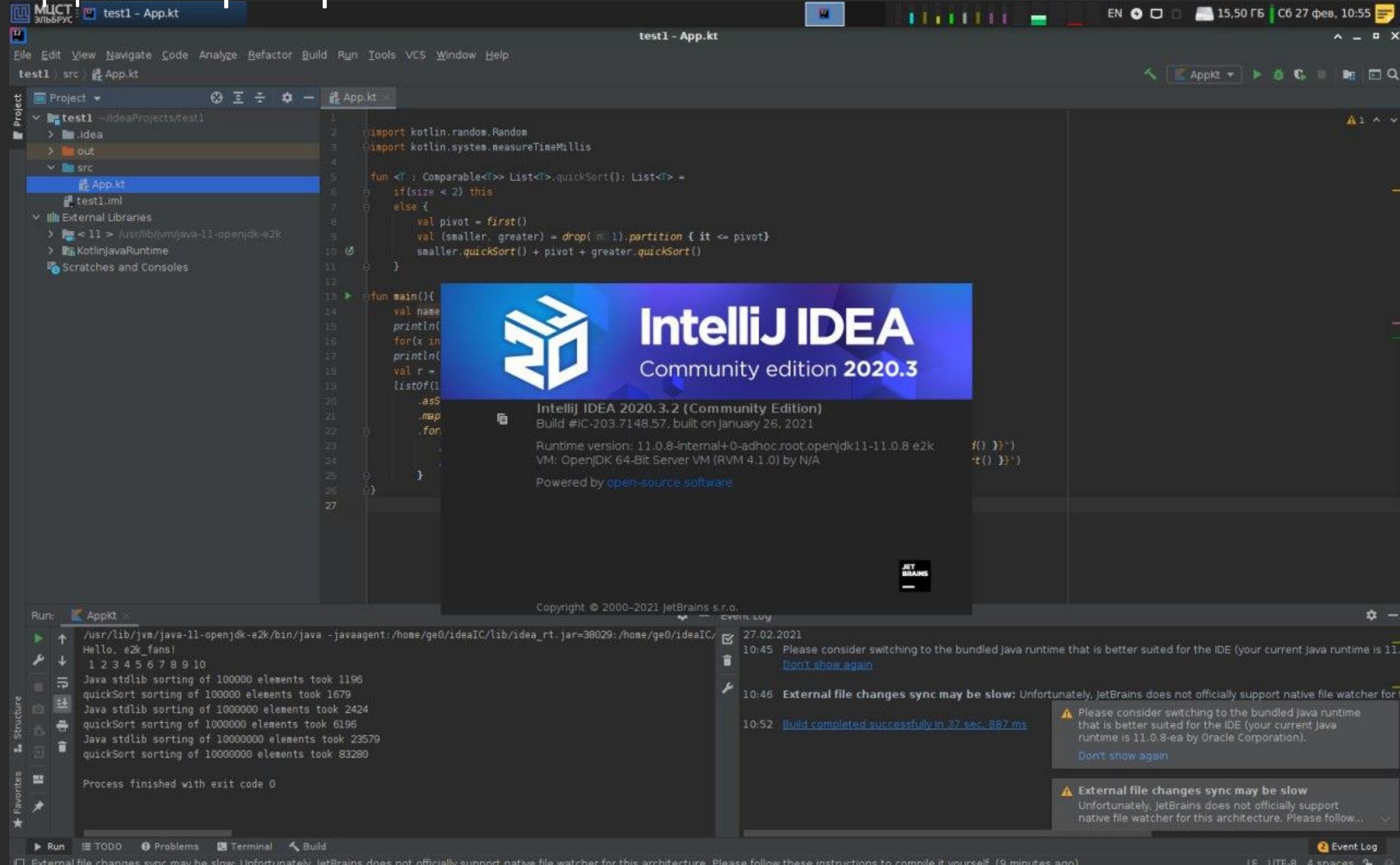
lcc: "/export/home/entityfx/anybench/sources/whets.c", строка 409: предупреждение:
    функция "start_time" объявлена неявно
    [-Wimplicit-function-declaration]
```

File: /export/home/entityfx/anybench/sources/whets.c Unix (LF) UTF-8 Line 5, Column 40 Insert Read/Write default

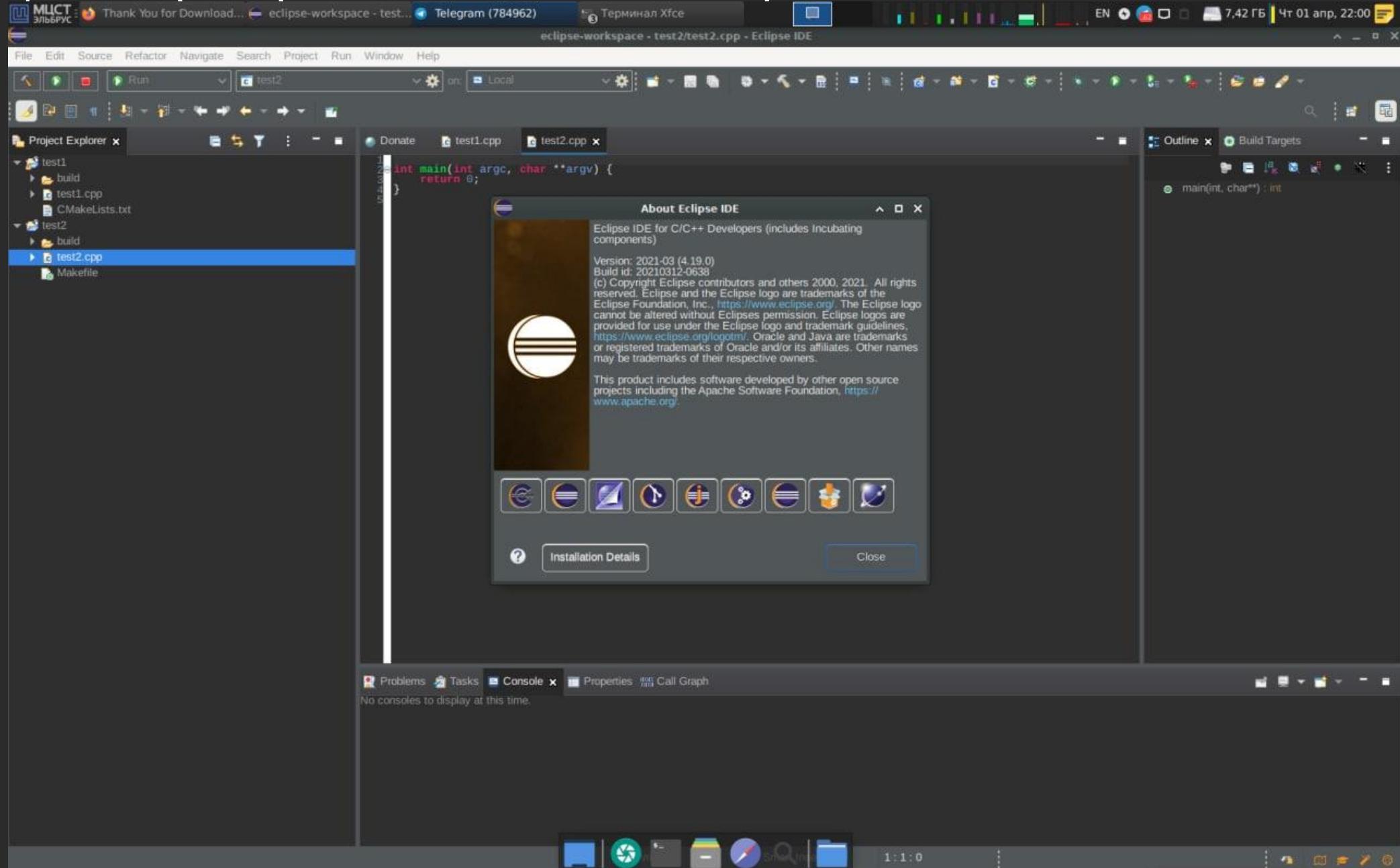
Средства разработки. IDE Qt Creator.



Средства разработки. IDE IntelliJ IDEA+.



Средства разработки. IDE Eclipse.



Технология Безопасных Вычислений

Процессор Эльбрус. Технологии безопасных вычислений

В обычном режиме работы процессор Эльбрус, как и другие процессоры (x86, ARM, ...), использует для обращения к памяти адреса - числа, хранящиеся в 32- или 64-битных регистрах. Эти числа, с точки зрения процессора, не отличаются от любых других значений в памяти. В режиме ТБВ Эльбрус для обращения в память использует только дескрипторы — 128-битные структуры, которые хранят в себе: адрес обращения, базу объекта и размер объекта.

Для программиста на языке высокого уровня переход в режим ТБВ несложен. При переносе программы в режим ТБВ изменится размер указателя с 32 или 64 на 128 битов. Прочие операции с указателями в режиме ТБВ будут, как и в обычном режиме, проводиться согласно стандарту языка.

ОБЫЧНЫЙ РЕЖИМ

адрес

32 / 64 бита

РЕЖИМ ТБВ

адрес

база

размер

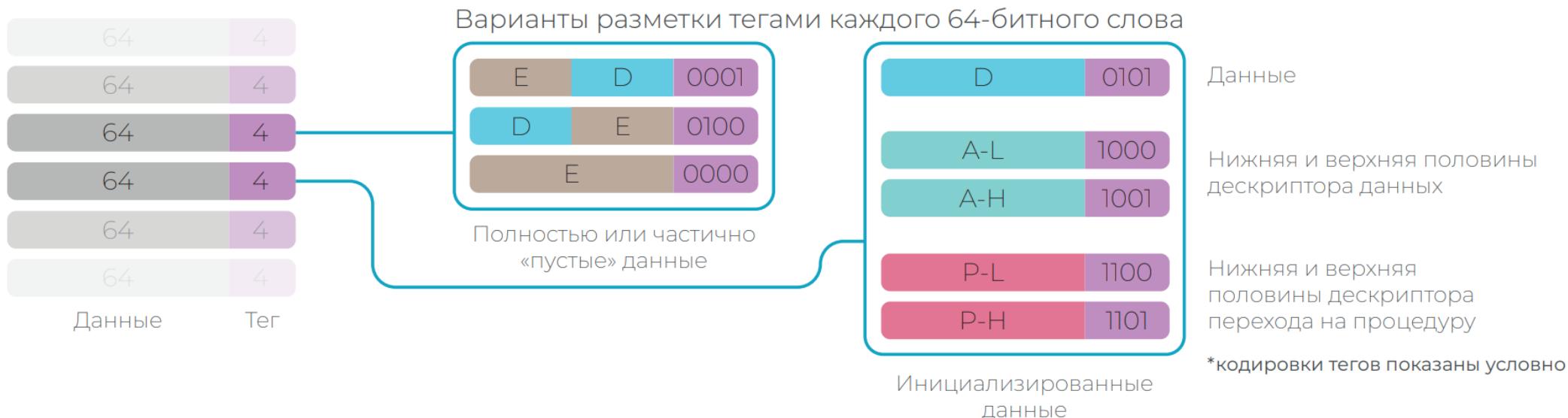
128 битов

Процессор Эльбрус. Технологии безопасных вычислений

Для каждого выровненного 64-разрядного слова, процессор Эльбрус хранит в памяти 4-битный тег — «невидимое» значение, которое описывает тип данных, хранящихся в этом 64-разрядном слове. Типом может быть: пустое (неинициализированное) значение, данные, нижняя часть 128-битного дескриптора, верхняя часть 128-битного дескриптора (отдельно для работы с данными, отдельно для вызова процедуры). Как «пустые», могут быть отмечены отдельно верхние или нижние 32 бита данных. В других процессорах аппаратная поддержка тегов отсутствует, а программная — несёт очень высокие накладные расходы.

Для ТБВ используются типы:

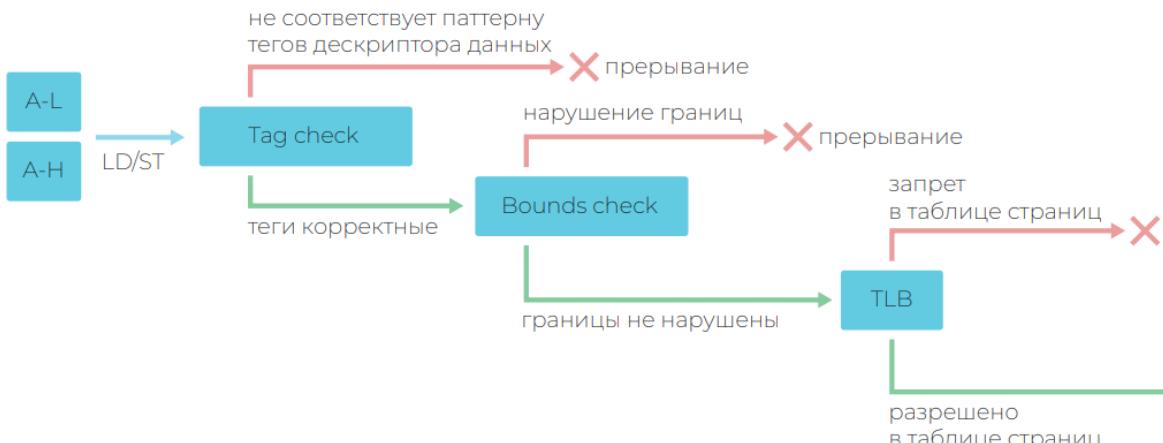
- числовые данные;
- указатель (дескриптор);
- неинициализированные данные.



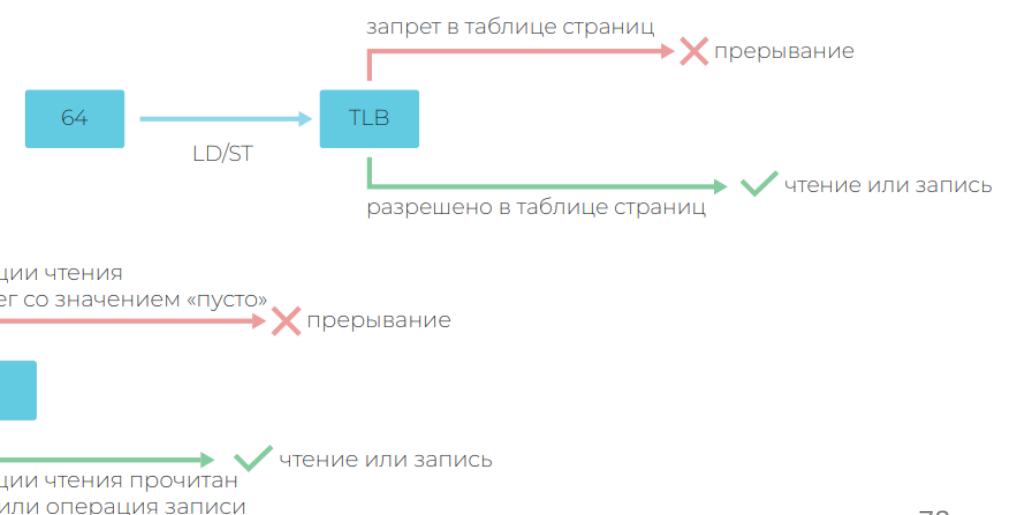
Процессор Эльбрус. Технологии безопасных вычислений

В обычном режиме работы процессора Эльбрус, как и у других процессоров (x86, ARM, ...), обращения в память контролируются только через таблицу страниц. В защищённом режиме, при обращении к памяти через дескриптор, процессор дополнительно проверяет, находится ли адрес обращения в границах, предписанных дескриптором. При нарушении границ вызывается прерывание. При вызове процедуры проверяется, что дескриптор предназначен для перехода, а не описывает область памяти с данными. Контроль границ происходит до передачи запроса в кэш-память, поэтому в режиме ТБВ исключается эксплуатация уязвимостей типа Spectre. За счёт системы тегов исключена возможность «напрямую» изменять части дескриптора, что гарантирует целостность системы контроля границ объектов.

РЕЖИМ ТБВ



ОБЫЧНЫЙ РЕЖИМ



Процессор Эльбрус. Технологии безопасных вычислений

ТБВ решает задачи:

- Обнаружение программных ошибок следующего типа:
 - Переполнение буфера Использование неинициализированных данных
 - Обращение к уже освобождённому объекту в памяти (по зависшей ссылке)
 - Защита от утечек по побочным каналам памяти типа Spectre
- Предоставление разработчикам информационных систем новых способов разграничения доступа, более эффективных, чем имеющееся сегодня разделение адресных пространств процессов.

При компиляции используется флаг: **-m128**

Ограничения:

Для компиляции в РБВ программа на С/С++:

- не должна содержать преобразование из целого в указатель;
- не должна закладываться на `sizeof(void*) == sizeof(long)`;
- не должна закладываться на `sizeof(void*) == sizeof(double)`;
- не может содержать отдельных объектов размером более 2^{32} - это ограничение является временным для текущей реализации.

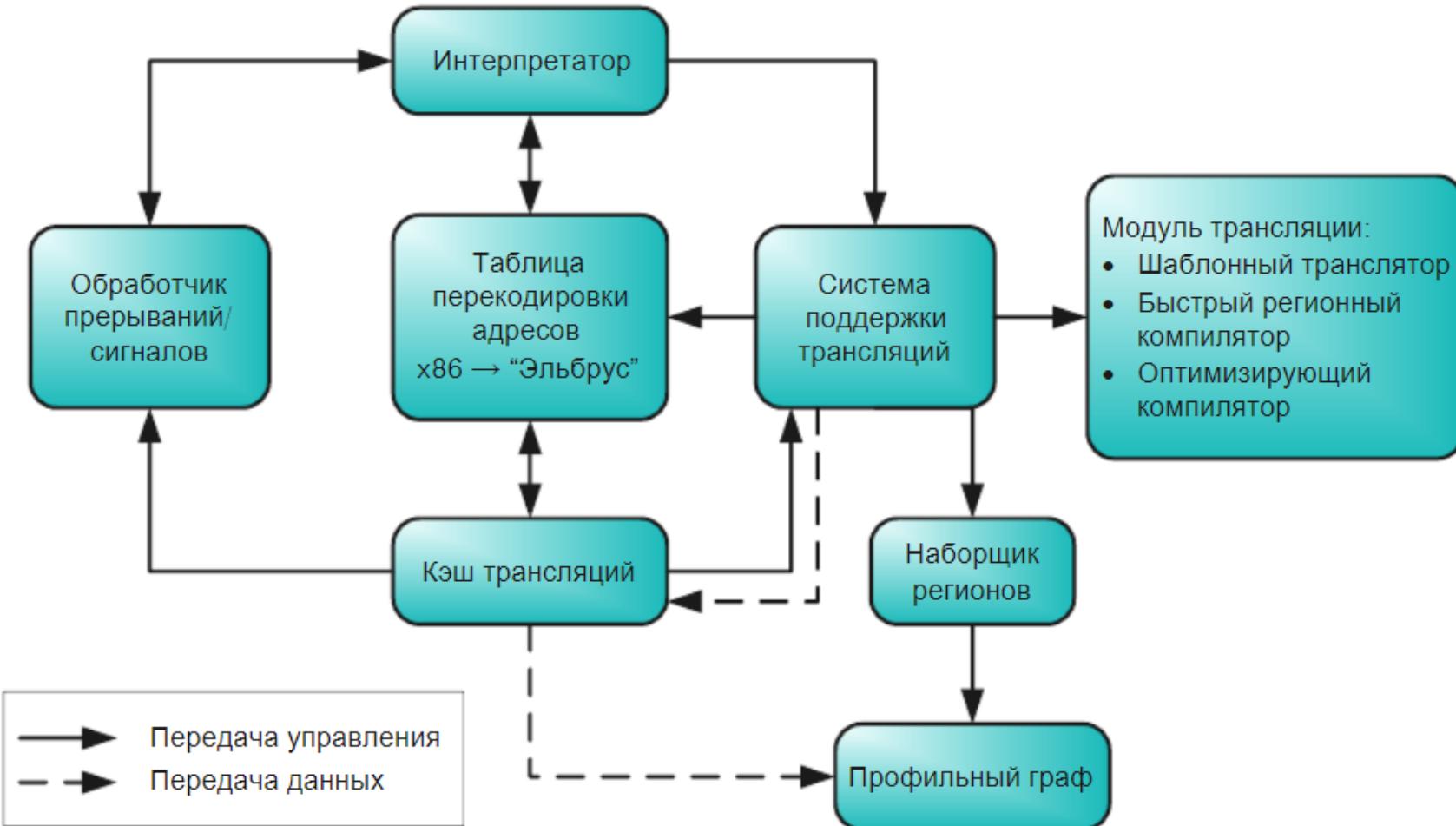
В процессе исполнения в ТБВ программа получит исключительную ситуацию:

- при обращении по указателю за пределами, указанными в дескрипторе `[base, base+size]`;
- при использовании данных, прочтенных из неинициализированной памяти;
- при попытке разыменования некорректного дескриптора (попытка неявного преобразования целого в указатель через память).

Двоичная трансляция

Lintel, RTC

Двоичная трансляция на процессорах Эльбрус

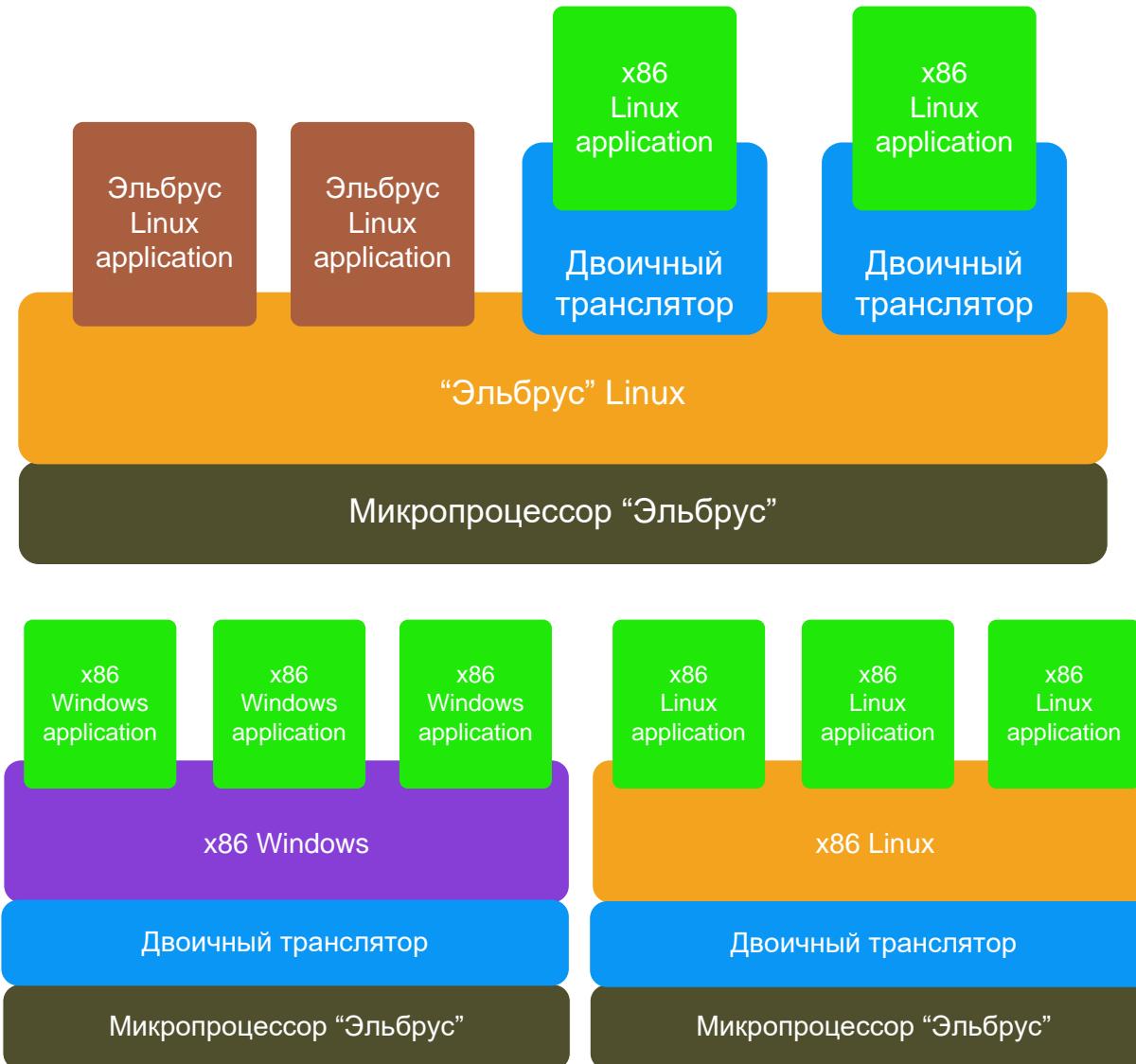


Двоичная трансляция на процессорах Эльбрус

Двоичный транслятор сделан в виде набора из трёх трансляторов и состоит из трех уровней кодогенерации:

- Шаблонный, который реализует пошаговую трансляцию: берет одну инструкцию Интела, синтезирует для нее соответствующий набор инструкций Эльбруса и исполняет. Это эффективно только для кода, который исполняется один раз.
- Быстрый регионарный, который берет уже не одну инструкцию, а линейный фрагмент и компилирует его, учитывая взаимосвязи между инструкциями. В скомпилированный код вставляются точки замера, которые показывают часто или редко исполняется код. Для часто исполняемого кода запускается третий уровень кодогенерации.
- Оптимизирующий уровень работает медленно и применяется для перекомпиляции горячих участков кода. Это тяжелый оптимизирующий компилятор, который выжимает из бинарной трансляции все, что возможно, и синтезирует эффективный код. По быстродействию он мало уступает прямому исполнению на Intel.

Реализации двоичной трансляции: RTC и Lintel



Двоичный транслятор. Утилита RTC.

Компонент [системы двоичной трансляции](#), известный как RTC, позволяет запускать на компьютере архитектуры Эльбрус под управлением операционной системы «Эльбрус Линукс» (или иной ОС семейства Linux) прикладные программы для Linux в машинных кодах x86 или x86-64 — например, WINE, Visual Studio Code — без перекомпиляции из исходных текстов.

Трансляция проходит в режиме реального времени, «на лету», с адаптивной многопроходной оптимизацией, что в сочетании с аппаратными средствами поддержки трансляции, заложенными в архитектуру Эльбрус и обеспечивающими низкие накладные расходы, даёт высокую скорость работы гостевых приложений. При этом системные вызовы ядра Linux обрабатываются ядром хозяйской системы, что также снижает накладные расходы по сравнению с запуском целой системы x86 Linux через [транслятор уровня системы](#).

Двоичный транслятор. Запуск.

хост (хостовая операционная система) — ОС, под которой мы запускаем транслятор, в которую кладём образ гостевой системы и т.п. Имеет архитектуру e2k, в файловой системе лежат бинарники под e2k.

гость (гостевой образ, гостевой chroot) — операционное окружение, файловую систему которого (в виде подкаталога в файловой системе хоста) мы подсовываем транслятору и в которую rtc "делает chroot" при запуске. Имеет архитектуру x86 или x86_64, в файловой системе лежат бинарники такой же архитектуры. Ядро не требуется.

Запуск 32-разрядного окружения:

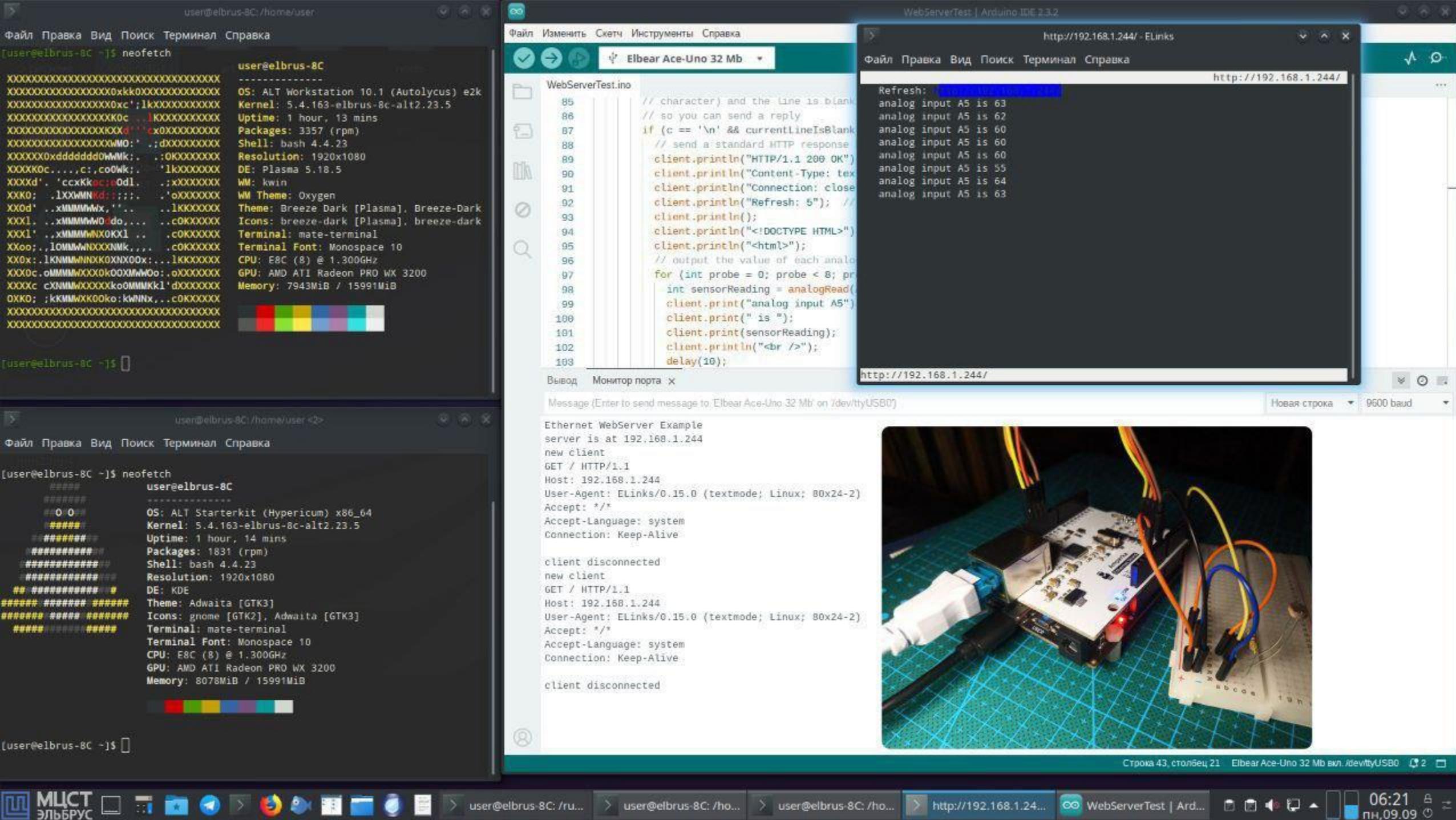
```
/opt/mcst/rtc/bin/ --path_prefix /opt/mcst/os_elbrus.3.0-rc36.x86/ -b $HOME -b /etc/passwd -b /etc/group -- /bin/bash
```

Запуск 64-разрядного окружения:

```
/opt/mcst/rtc/bin/rtc_opt_rel_p1_x64_ob --path_prefix /opt/mcst/os_elbrus.4.0-rc4.x86_64/ -b $HOME -b /etc/passwd  
-b /etc/group -- /bin/bash
```

Двоичный транслятор. Утилита RTC. Возможности.

Категория	Особенность / возможность	Примечание
система команд	Реализована поддержка базового набора инструкций x86 и x86-64, а также некоторых расширений, например SSE.	Конкретный набор поддерживаемых инструкций определяется возможностями аппаратуры того или иного процессора.
системные вызовы	Поддерживаются гостевые приложения, использующие библиотеку libc, совместимую с версией ядра Linux, работающего на хозяйской платформе.	Транслятор приложений при работе адаптирует системные вызовы приложения гостевой платформы в системные вызовы ядра ОС Эльбрус.
ioctl	Поддерживается проверенный набор ioctl-запросов, который может быть расширен по запросам пользователей.	Транслятор приложений при работе адаптирует структуры данных в гостевых запросах к виду, принятому в хозяйской системе.
файловая система	Статически скомпонованные программы могут быть запущены сами по себе. Динамически скомпонованные программы могут быть запущены в подходящем окружении, включающем как минимум все необходимые им динамически подгружаемые библиотеки.	Определить перечень необходимых гостевых библиотек можно программой ldd.
"	Гостевым окружением может служить набор файлов проинсталлированной обычным способом ОС Linux x86 — как скопированный в хозяйскую файловую систему, так и подключённый напрямую с накопителя от x86-компьютера или компьютера Эльбрус под управлением транслятора системы (Lintel).	На практике в той или иной степени могут быть работоспособны программы из состава Debian, Ubuntu, CentOS, Fedora, OpenSUSE и других дистрибутивов GNU/Linux. Официально в качестве гостевого окружения поддерживается только ОС Эльбрус , так как гарантировать совместимость произвольно взятой ОС невозможно.
"	В гостевом окружении доступны каталоги /proc (процессы), /dev (устройства) и /sys (ядро) из хозяйской системы.	Перечень прорасываемых каталогов может быть расширен при помощи скрипта bind.sh — текущая версия транслятора позволяет задать до 64 директорий и файлов.

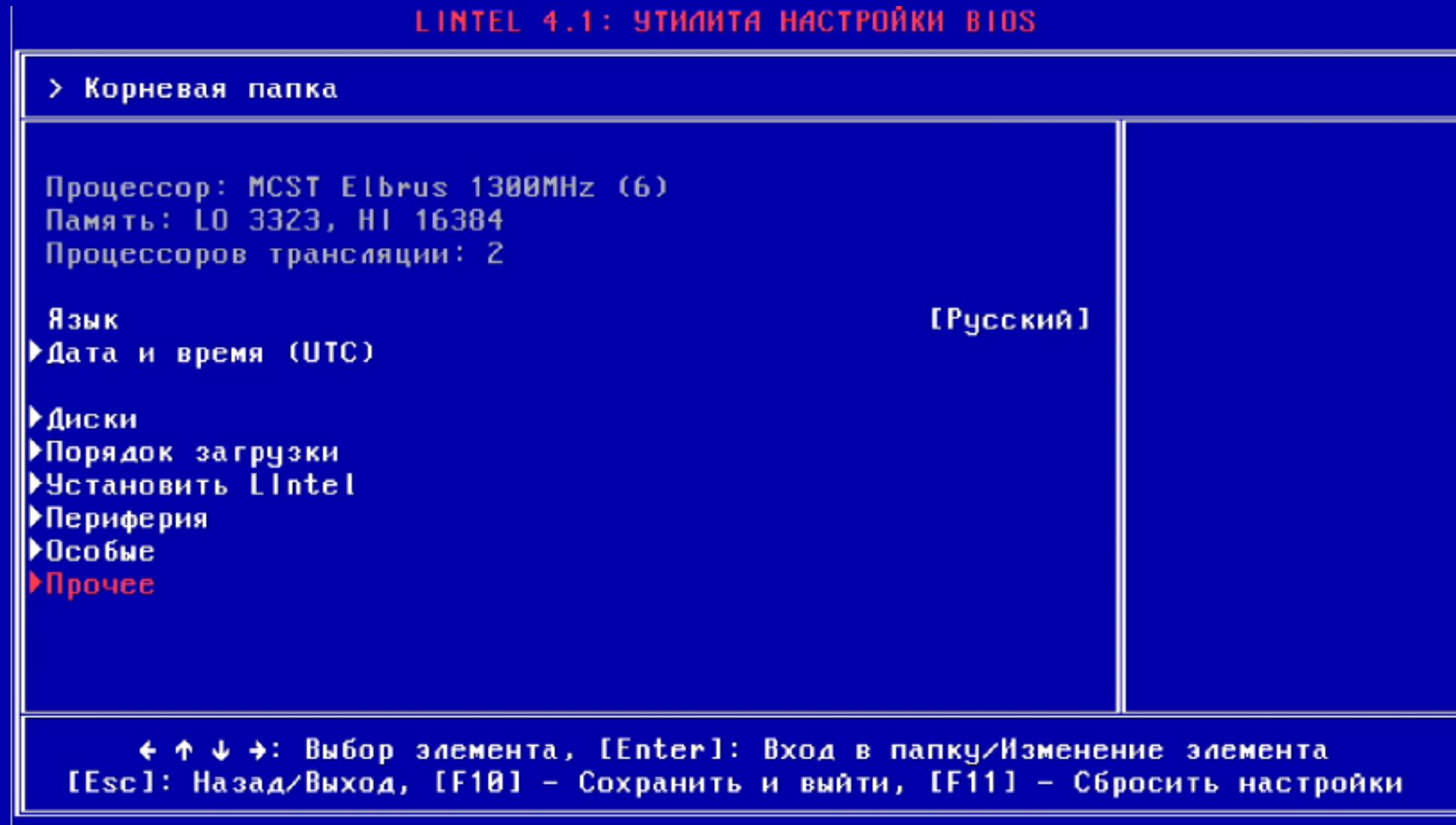


Двоичный транслятор. Lintel.

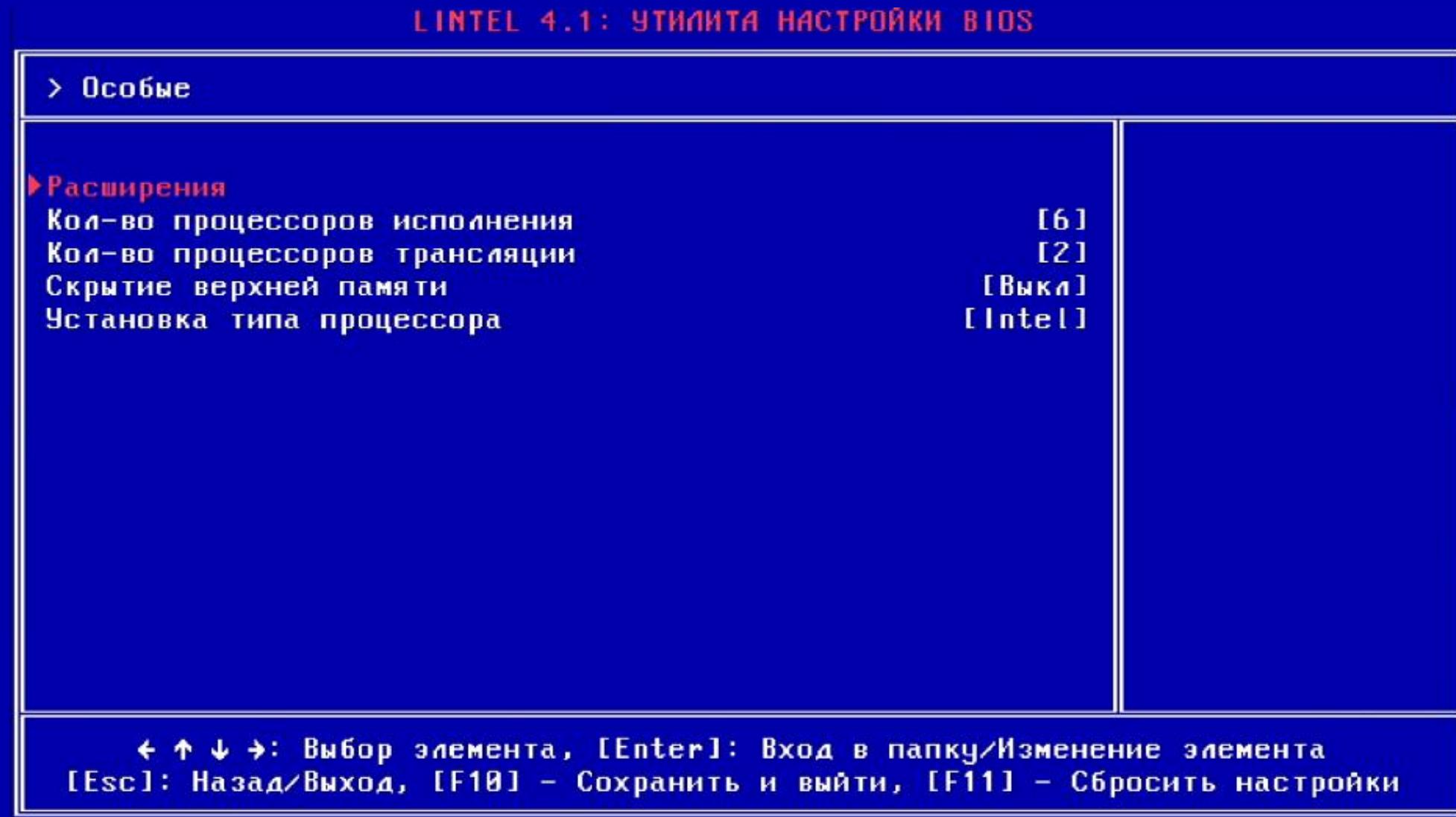
Компонент [системы двоичной трансляции](#), известный как **Lintel**, позволяет запустить на компьютере архитектуры Эльбрус операционную систему в машинных кодах x86 или x86-64 (amd64) — например, Microsoft Windows или Red Hat Enterprise Linux — без перекомпиляции из исходных текстов.

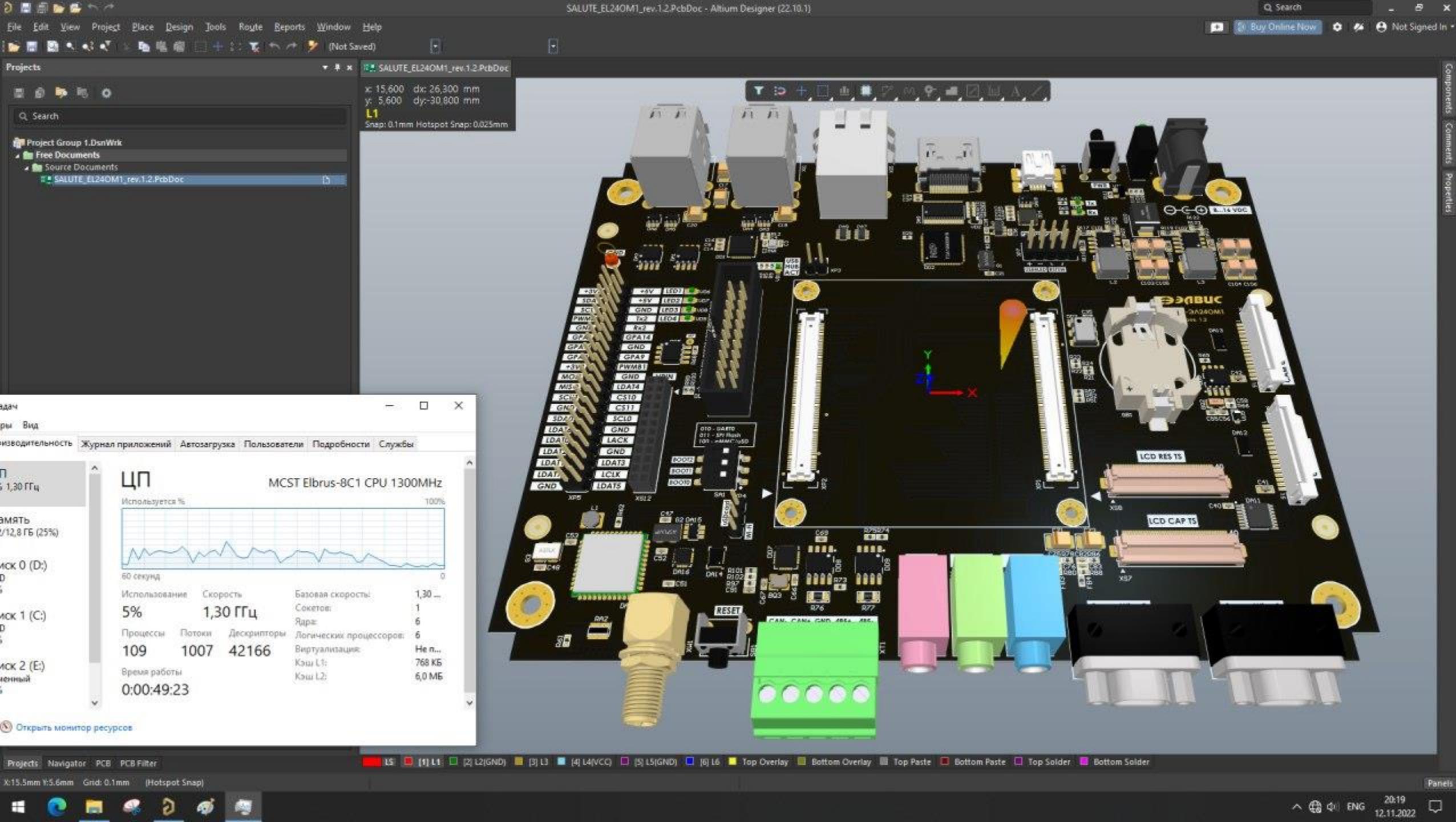
Трансляция проходит в режиме реального времени, «на лету», с адаптивной многопроходной оптимизацией, что в сочетании с аппаратными средствами поддержки трансляции, заложенными в архитектуру Эльбрус и обеспечивающими низкие накладные расходы, даёт высокую скорость работы гостевых систем. В отличие от [транслятора приложений](#), транслятор уровня системы создаёт наиболее полное подобие имитируемого x86-компьютера.

Lintel BIOS. Главное меню.



Lintel BIOS. Настройка ядер трансляции/исполнения.



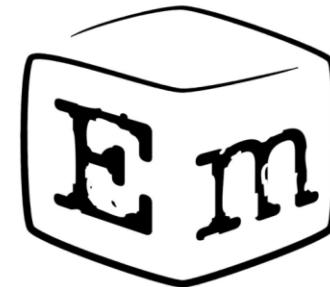


Операционные системы

Процессор Эльбрус. Поддерживаемые ОС.

Операционные системы, поддерживающие архитектуру e2k:

- ALT Linux (<https://www.basealt.ru/>) (@alt_linux)
- Astra Linux (<https://astralinux.ru/>) (@astralinux_chat)
- Эльбрус Linux: (http://mcst.ru/elbrus_os)
- РедОС
- ОСРВ БагрОС-4000
- ОСРВ Нейтрино
- ОС Лотос
- EmBox



Пакеты:

- ОС Эльбрус 6.0.1 > 2500
- Astra Linux SE 1.6 > 9800
- Alt Sisyphus > 15500



Версии ядер Linux: **3.14, 4.9, 4.19, 5.4, 5.10, 6.1**

Процессор Эльбрус. Эльбрус Линукс.

Операционная система **Эльбрус Линукс**, также известная как **ОС Эльбрус** (OSL), создана специалистами компании АО МЦСТ - разработчика архитектуры Эльбрус и компьютерной техники на её основе. Относится к семейству GNU/Linux, сочетаая ядро Linux и прикладные программы GNU, а также более 2000 программных пакетов. Является полностью собственной разработкой, то есть не копирует ни один другой дистрибутив операционной системы, хотя и включает в себя технические решения Debian.

Коммерческий дистрибутив выпускается для компьютеров архитектуры Эльбрус и SPARC (МЦСТ-R) и поставляется вместе с такими компьютерами или отдельно, по договору поставки. Также выпускается «Эльбрус Линукс x86» для компьютеров архитектуры x86-64 - свободно распространяемое дополнение к основному дистрибутиву, которое может использоваться как самостоятельно, так и в сочетании с компьютерами Эльбрус при кросс-компиляции или двоичной трансляции.

Отличительные особенности «Эльбрус Линукс»:

- собственная пакетная база, пополняемая напрямую от разработчиков открытого программного обеспечения;
- монолитные пакеты: каждый пакет включает в себя запускаемые программы, файлы данных, средства для разработчика, - поэтому 1 пакет из состава «Эльбрус Линукс» может быть равносителен 5–20 пакетам в других дистрибутивах;
- собственная система сборки пакетов ([PDK](#));
- формат пакетов deb и программы управления пакетами dpkg и apt.



Документы



Твд 2.5 TB



Твд 1.1 GB



Корзина



Файловый
система

bin - 86Box 4.0

Action View Media Tools Help

||| ○ ○ ○ ○ ○ ○

Award Modular BIOS v4.51PG, An Energy Star Ally
Copyright (C) 1984-98, Award Software, Inc.

ASUS P5A ACPI BIOS Revision 1811 Beta 005

Pentium-MMX 166MHz Processor
Memory Test : 32768K OK

Award Plug and Play BIOS Extension v1.8A
Initialize Plug and Play Cards...
PnP Init Completed

Trend ChipAwayVirus(B) On Guard

Detecting HDD Primary Master ... None
Detecting HDD Primary Slave ... None
Detecting HDD Secondary Master... None
Detecting HDD Secondary Slave ... None

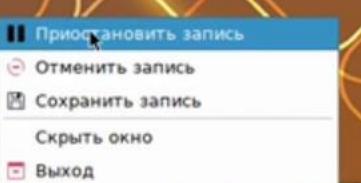
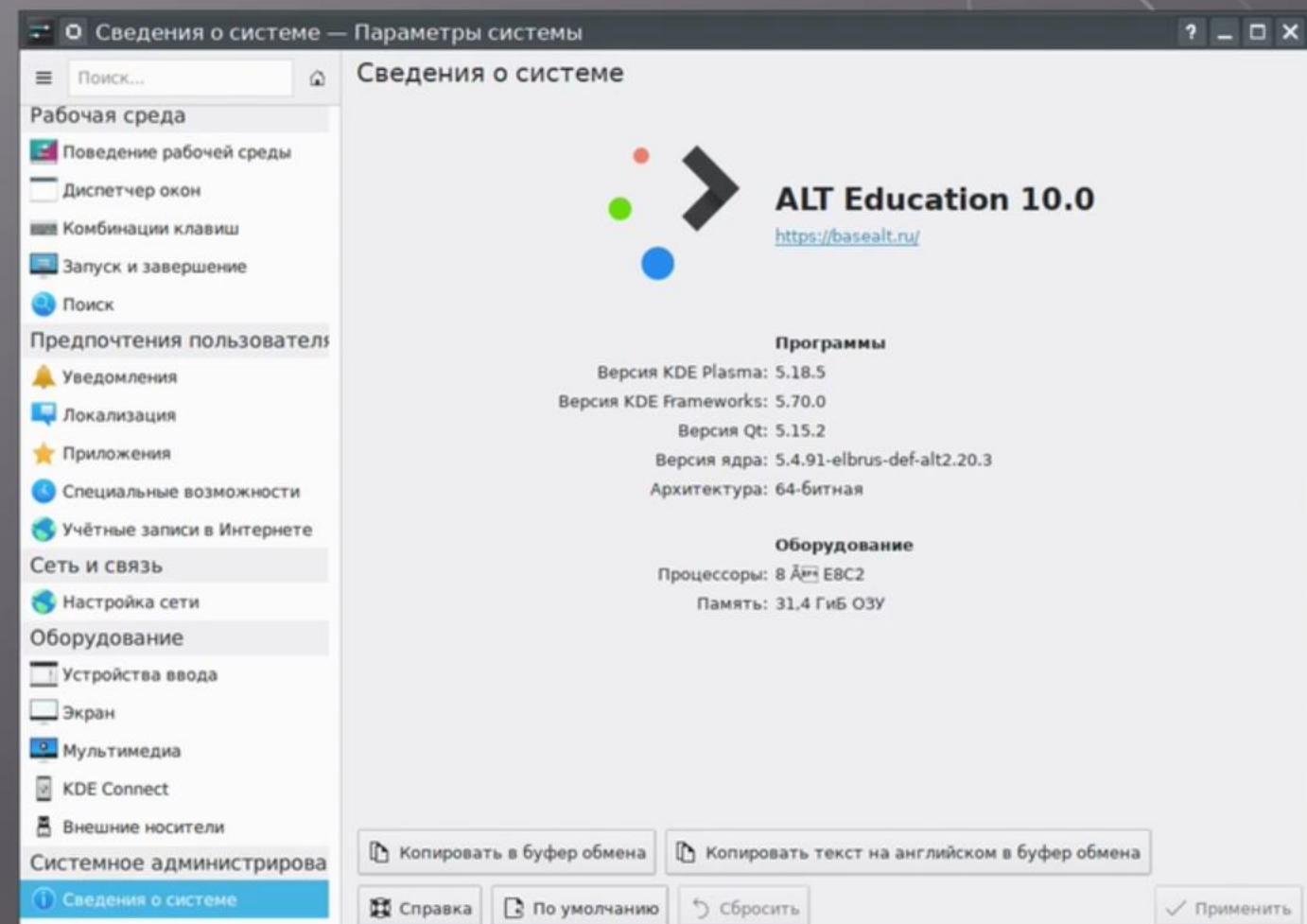
CMOS checksum error - Defaults loaded

Press F1 to continue, DEL to enter SETUP
05/02/2002-ALADDIN-{{PSA}}





о системе



Прикладное ПО

Процессор Эльбруса. Прикладное ПО.

Версия ОС	8.x	7.x	6.x	5.0	4.1	4.0	3.1	3.0
Mesa	21.3.9	21.3.9	20.3.5	19.3.5	17.2.8	17.2.8	17.0.7	17.0.7
Mate	1.24.1	1.24.1	1.24.1	-	-	-	-	-
Xfce	4.18.0	4.16.0	4.14.1	4.14.1	4.12.0	4.12.0	4.12.0	4.12.0
LibreOffice	6.3.0.4	6.3.0.4	6.3.0.4	5.2.1.2	5.2.1.2	5.2.1.2	5.2.1.2	5.2.1.2
Leafpad	0.8.17	0.8.17	0.8.17	0.8.17	0.8.17	0.8.17	0.8.17	0.8.17
Nano	4.9.2	4.9.2	4.9.2	4.9.2	2.8.7	2.8.7	2.8.7	2.8.7
Vim	9.0.1672	8.2.4788	8.2.0716	8.2.0716	8.0	8.0	8.0	8.0
Blender	3.4.1	2.93.5	2.80	2.80	-	-	-	-
GIMP	2.99.10	2.10.22	2.10.18	2.10.18	2.8.18	2.8.18	2.8.18	2.8.18
Inkscape	1.2.2	1.0.1	1.0.1	0.92.4	-	-	-	-
OBS Studio	20.1.3	20.1.3	20.1.3	-	-	-	20.1.3	20.1.3
VLC	3.0.14	3.0.14	3.0.14	3.0.8	2.2.4	2.2.4	2.2.4	2.2.4
Firefox	91.13.0	91.13.0	68.12.0	52.9.0	52.9.0	52.9.0	52.6.0	52.6.0
Thunderbird	91.13.1	91.13.1	52.9.1	52.9.1	52.9.1	52.9.1	52.6.0	52.6.0
Midnight Commander	4.8.23	4.8.23	4.8.23	4.8.23	4.8.23	4.8.23	4.7.0.8	4.7.0.8
MongoDB	3.6.13	3.6.13	3.6.13	3.6.13	3.6.13	3.6.13	-	-
MySQL	5.7.24	5.7.24	5.7.24	5.7.24	5.7.24	5.7.24	5.7.22	5.7.22
PostgreSQL	13.5	13.5	11.11	11.5	9.6.8	9.6.8	9.6.8	9.6.8
NGINX	1.18.0	1.18.0	1.18.0	1.12.0	1.12.0	1.12.0	1.12.0	1.12.0

Octane 2.0 JavaScript Benchmark - Mozilla Firefox

Octane Score: 2268

Richards	2478	Deltablue	4622	Crypto	4091	Raytrace	6105
Core language features		Core language features		Bit & Math operations		Core language features	
EarleyBoyer	3585	Regexp	290	Splay	885	SplayLatency	474
Memory & GC		Strings & arrays		Memory & GC		GC latency	
NavierStokes	3692	pdf.js	937	Mandrel	1874	MandrelLatency	1629
Strings & arrays		Strings & arrays		Virtual machine		Compiler latency	
GB Emulator	2598	CodeLoad	3266	Box2DWeb	3091	zlib	9356
Virtual machine		Loading & Parsing		Bit & Math operations		asm.js	
TypeScript	3453	Virtual machine & GC					

The final score is the geometric mean of the single scores. We suggest to restart the browser before repeating the test.

МЦСТ ЭЛЬБРУС

Избранное - (751005)

Octane 2.0 JavaScript Benchmark - Mozilla Firefox

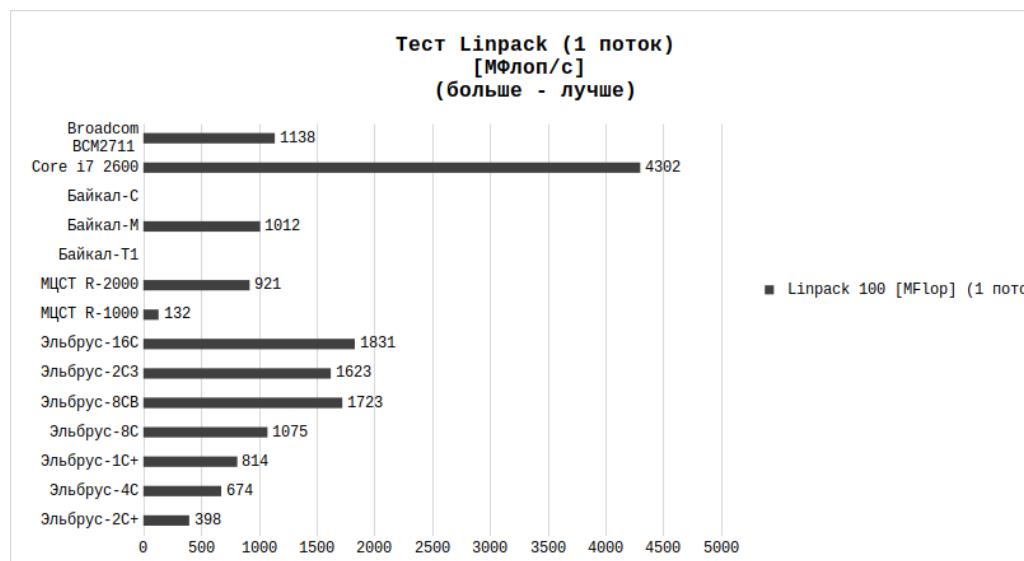
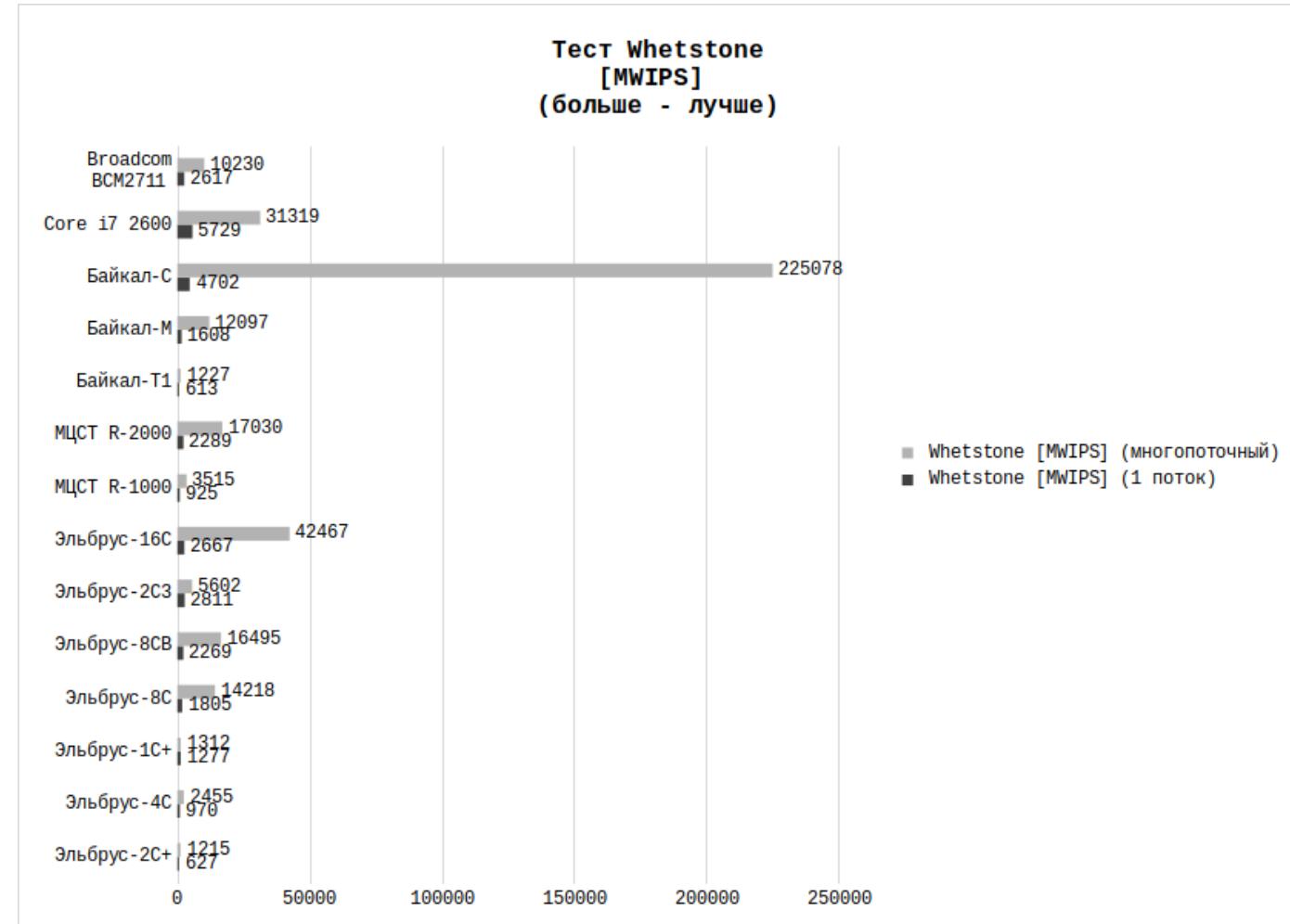
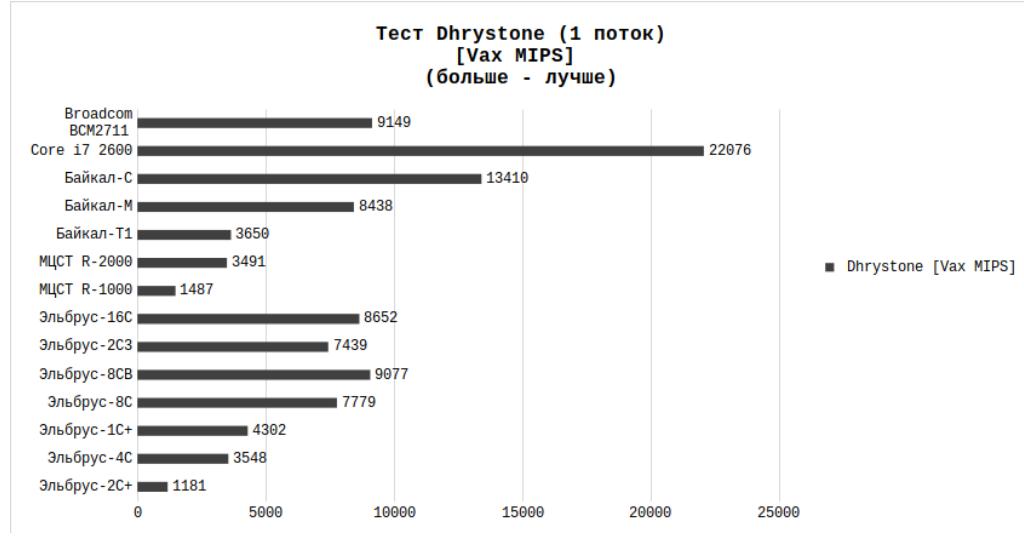
user@elbrus-8C:/home/user

user@elbrus-8C:/home/user >2>

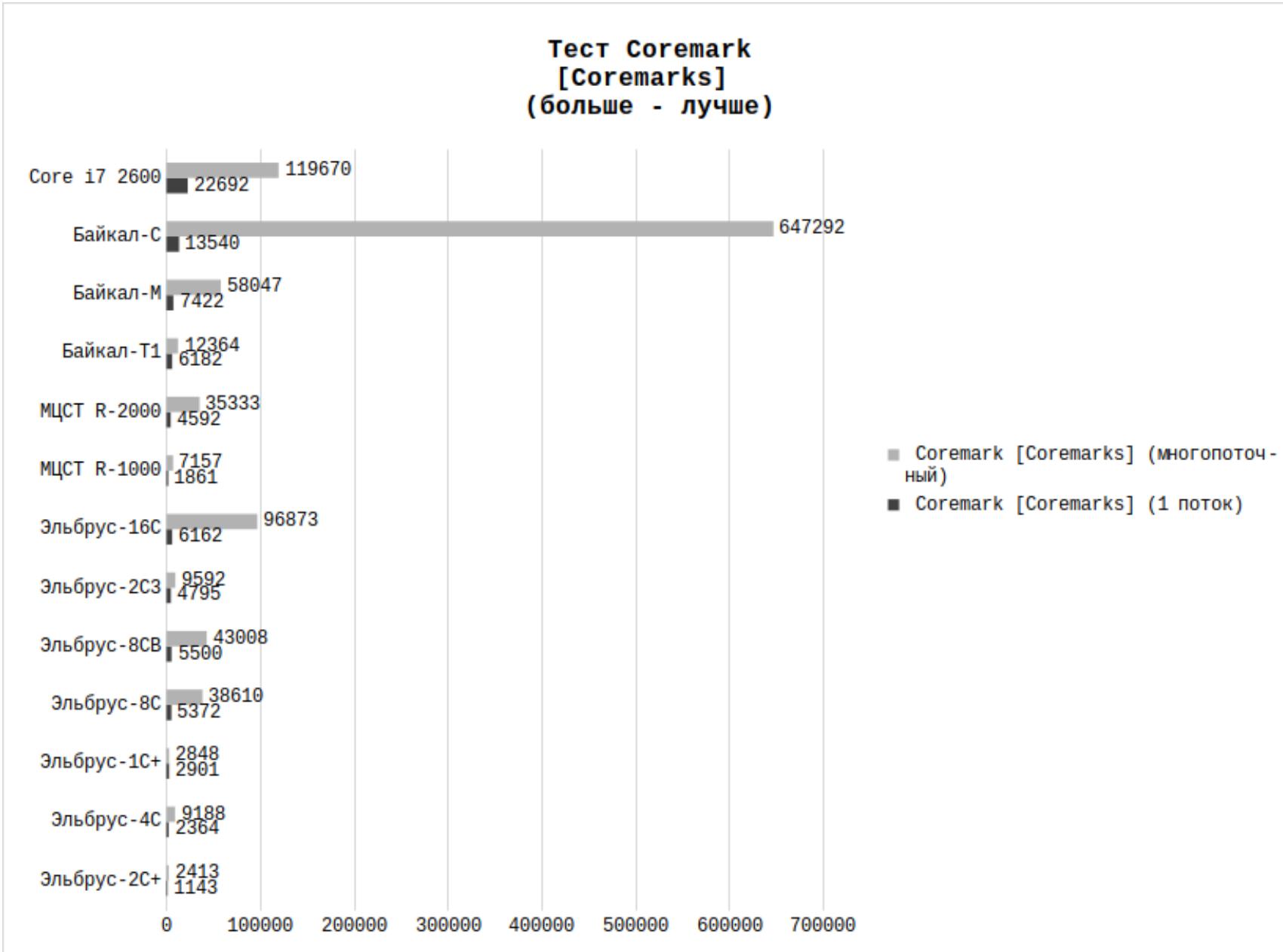
22:07 чт, 16.03

Бенчмарки процессоров Эльбрус

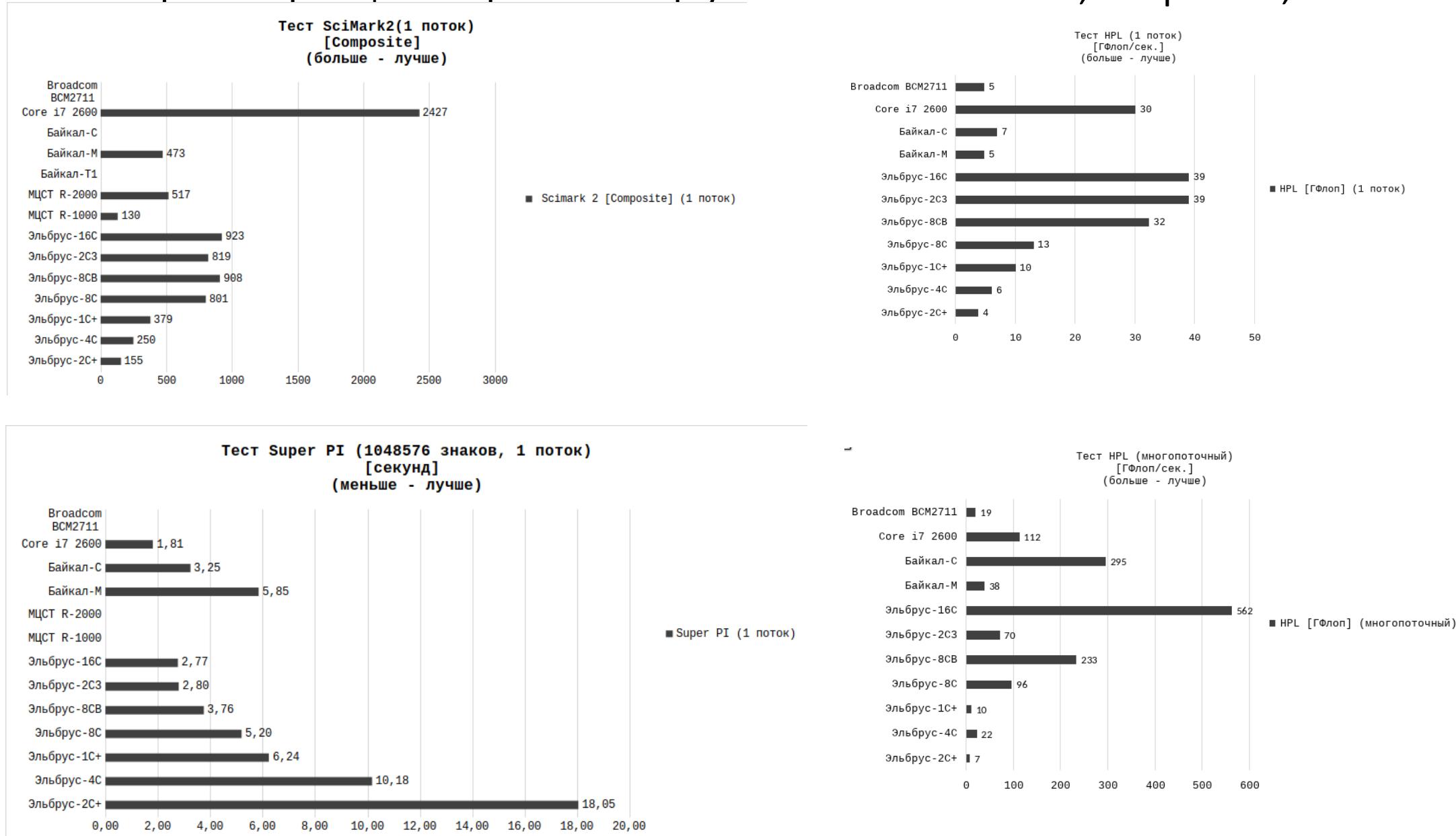
Бенчмарки Эльбруса. Тест Dhystone, Whetstone, Linpack 100



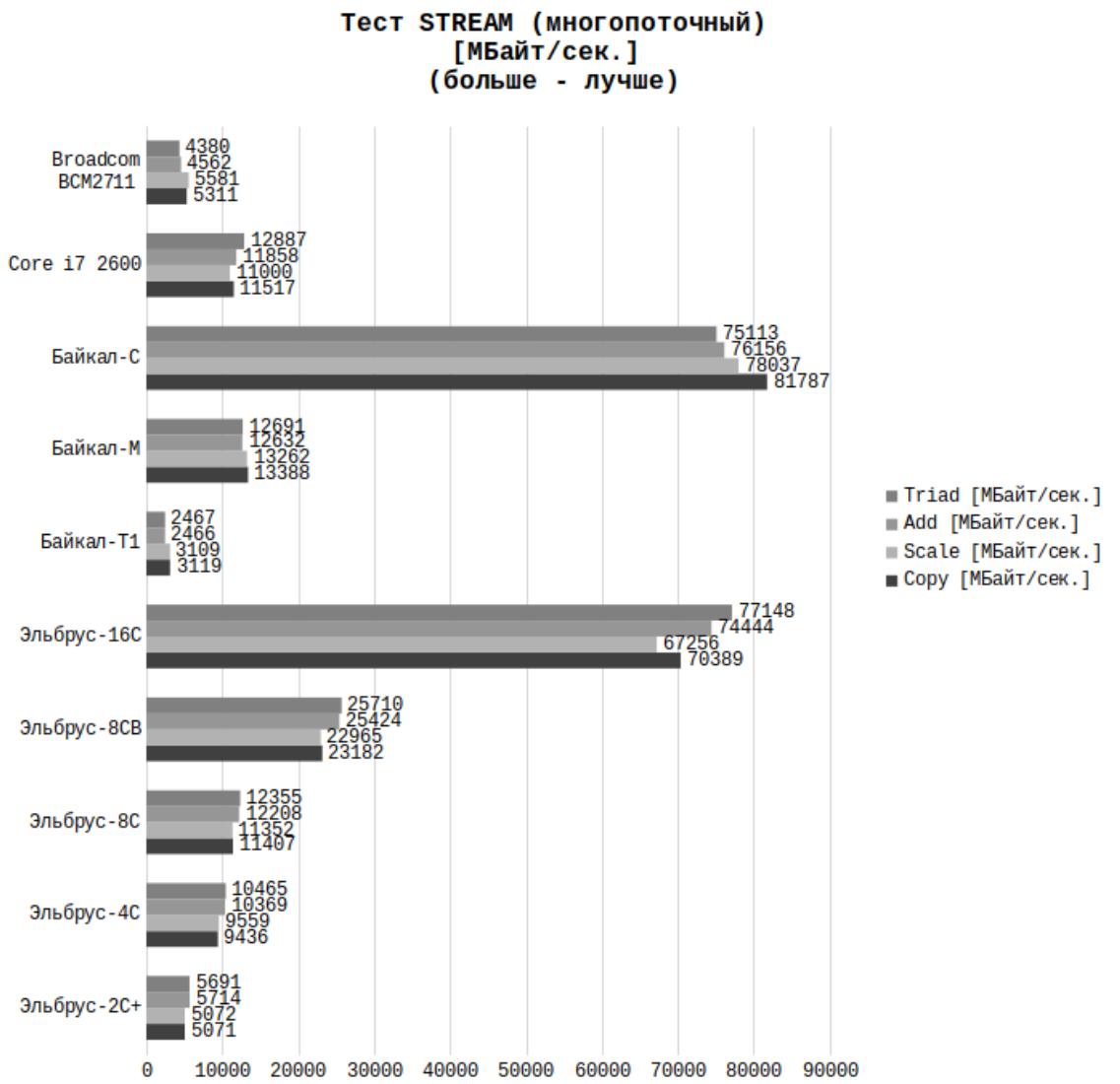
Бенчмарки процессора Эльбрус. Тест Coremark.



Бенчмарки процессора Эльбрус. Тест SciMark, SuperPi, HPL.



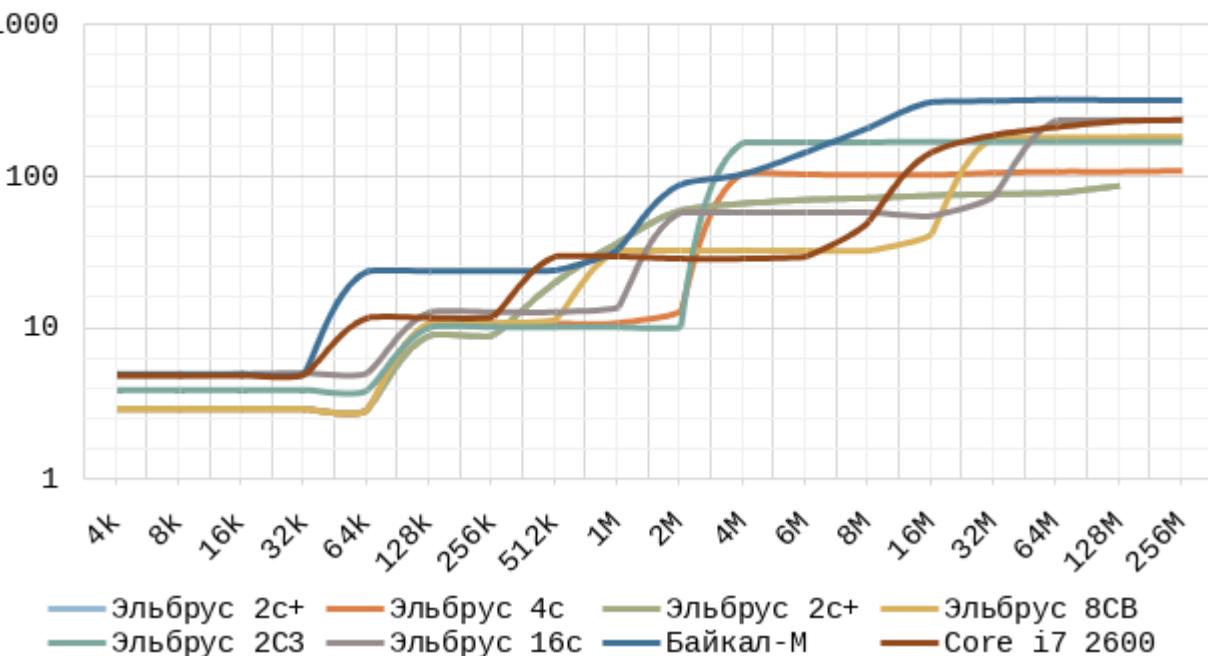
Бенчмарки процессора Эльбрус. Тест производительности ОЗУ.



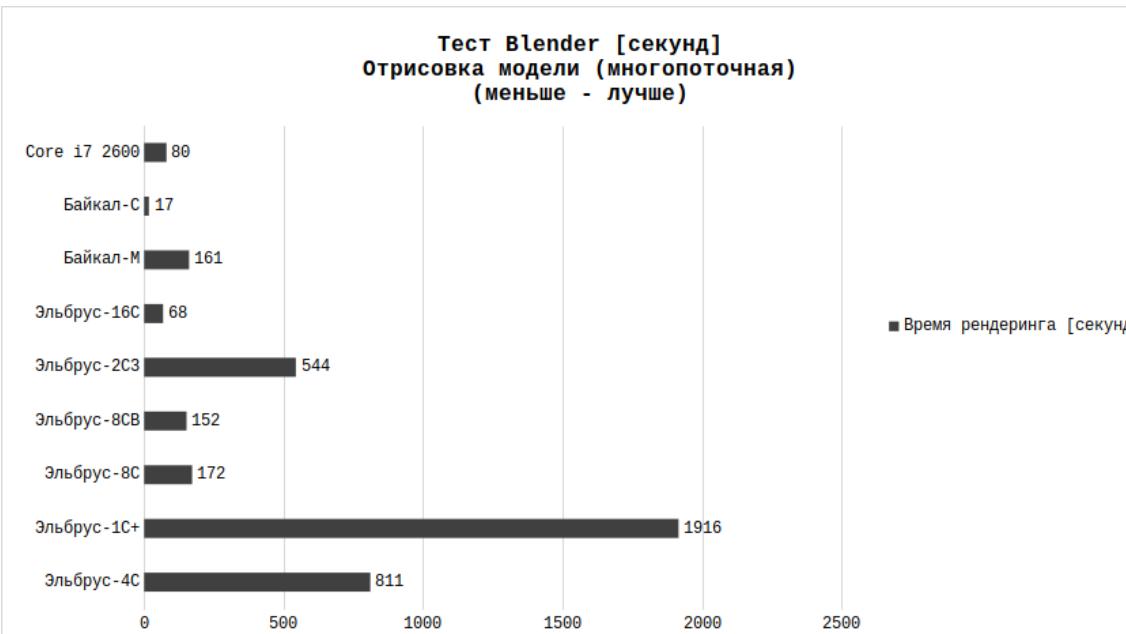
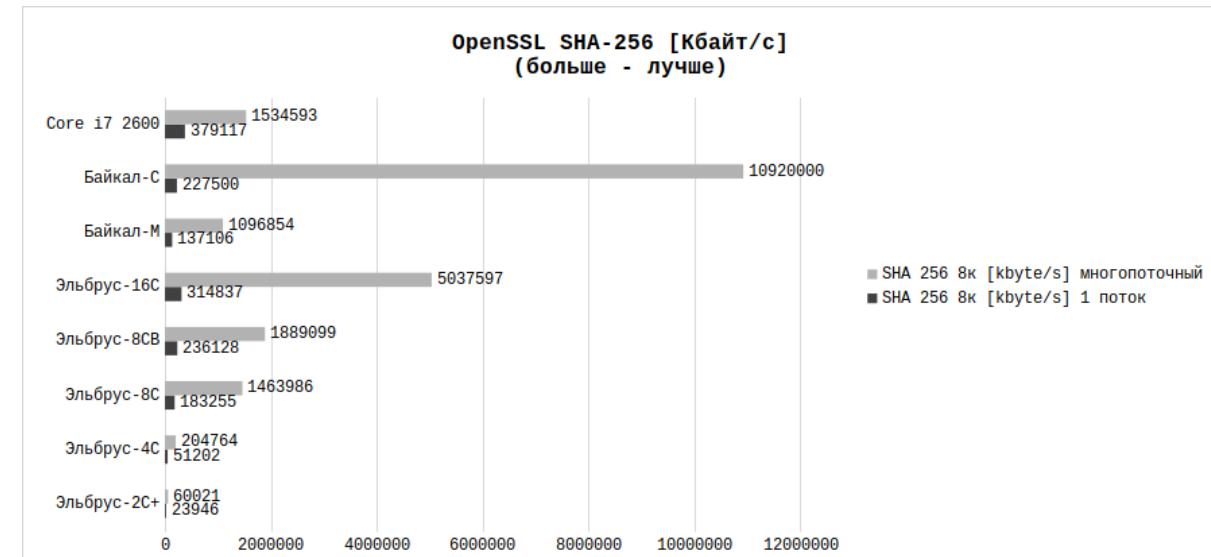
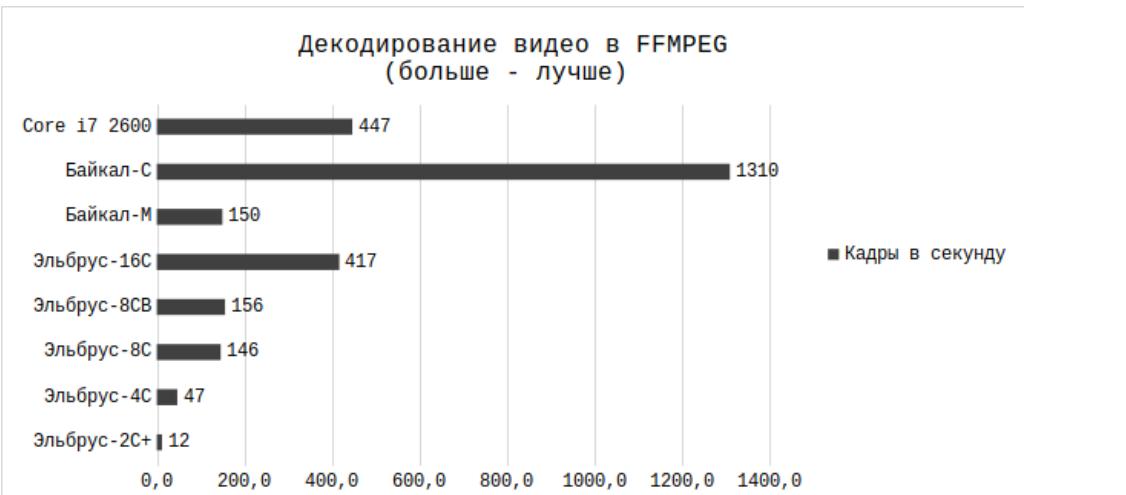
Результаты теста TLB показывают, что задержки доступа в память у процессоров Эльбрус составляют:

- 3 така в кеш 1 уровня;
- 11 тактов в кеш 2 уровня;
- 33 такта в кеш 3 уровня (при наличии данного уеша);
- от 80 до 200 тактов при доступе к ОЗУ.

Тест TLB со случайным доступом, размер страницы 4 кБайт
(1 поток)
[Задержка в тактах]
(меньше - лучше)



Бенчмарки процессора Эльбрус. Тест прикладным ПО.



Бенчмарки процессора Эльбрус. Тесты браузерного JS

Octane

Cpu	Result (баллы)
Intel Core i7 2600	23321
AMD A6-3650	11741
Intel Pentium 4 2800	3387
Elbrus 8C (rtc x86 32bit)	2815
Elbrus 8C	2102
Elbrus 1C+	739

Kraken Benchmark

Cpu	Result (ms)
Elbrus 8C	10493.4
Elbrus 8CB RTC x86	9567.5
Elbrus 8CB	8714.2
Intel Pentium 4 2800	9486.6
AMD A6-3650	3052.5
Intel Core i7 2600 (3.4 GHz)	1456.8

Sunspider

Cpu	Result (ms)
Elbrus 8C	3059.8
Elbrus 8CB	2394.6
Intel Pentium 4 2800	1295.5
AMD A6-3650	485.6
Intel Core i7 2600 (3.4 GHz)	242.9

Бенчмарки Эльбруса. Производительность ЯП.

- Java: Хорошо
- JS: Средняя
- C# Mono: Средняя
- Python: Ниже среднего
- PHP: Хорошо
- Lua: Ниже среднего

Бенчмарки Эльбруса. Производительность ЯП.

- <https://www.altlinux.org/Эльбрус/тесты/результаты>
- <https://github.com/EntityFX/anybench>
- <https://github.com/EntityFX/EntityFX-Bench>
- <https://docs.google.com/spreadsheets/d/1fBPxWAP13WwT-rOoqfseFl3vXRaSUSc4LU3ERzyegwg/edit>
- <https://7-cpu.com/>
- <https://github.com/jeffhammond/STREAM>
- <https://github.com/EntityFX/test-tlb>
- <https://github.com/pooler/cpuminer.git>
- <https://github.com/official-stockfish/Stockfish/pull/3425/files>
- <https://github.com/Fibonacci43/SuperPI>
- https://habr.com/ru/company/icl_services/blog/534296/
- https://habr.com/ru/company/icl_services/blog/501588/
- https://habr.com/ru/company/icl_services/blog/558564/
- http://ftp.altlinux.org/pub/people/mike/elbrus/docs/elbrus_prog/html/index.html
- http://www.mcst.ru/doc/book_121130.pdf
- <https://ce.mentality.rip/>
- <http://46.49.41.80/hint/>
- <https://github.com/ironss/hint-benchmark>

Бенчмарки Эльбруса. Игры.



DOOM 3



Re 3, Re:VC



Quake 2



Open Arena



Serious Engine



Source Engine



Source engine

Dagor engine

Open gothic

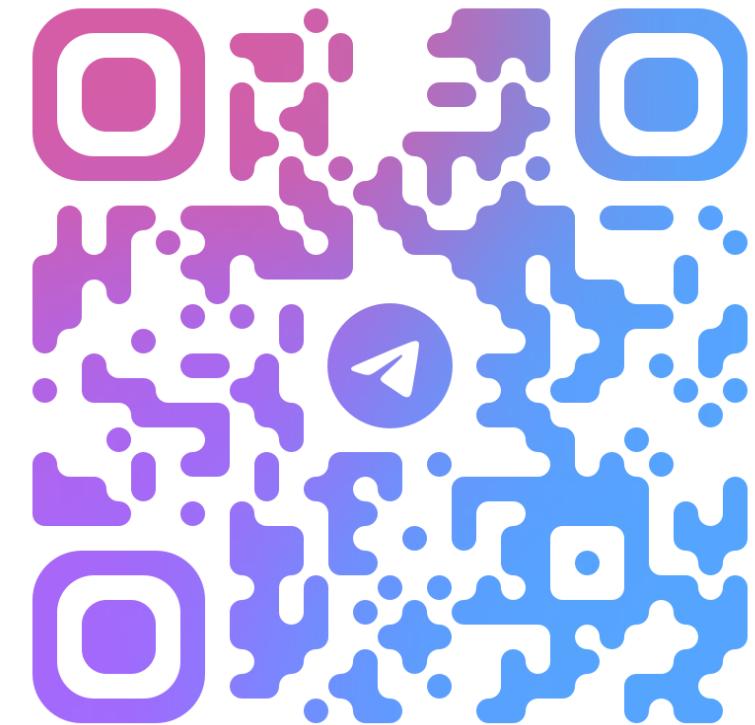
Open MW

Falltergeist

Kissak Strike

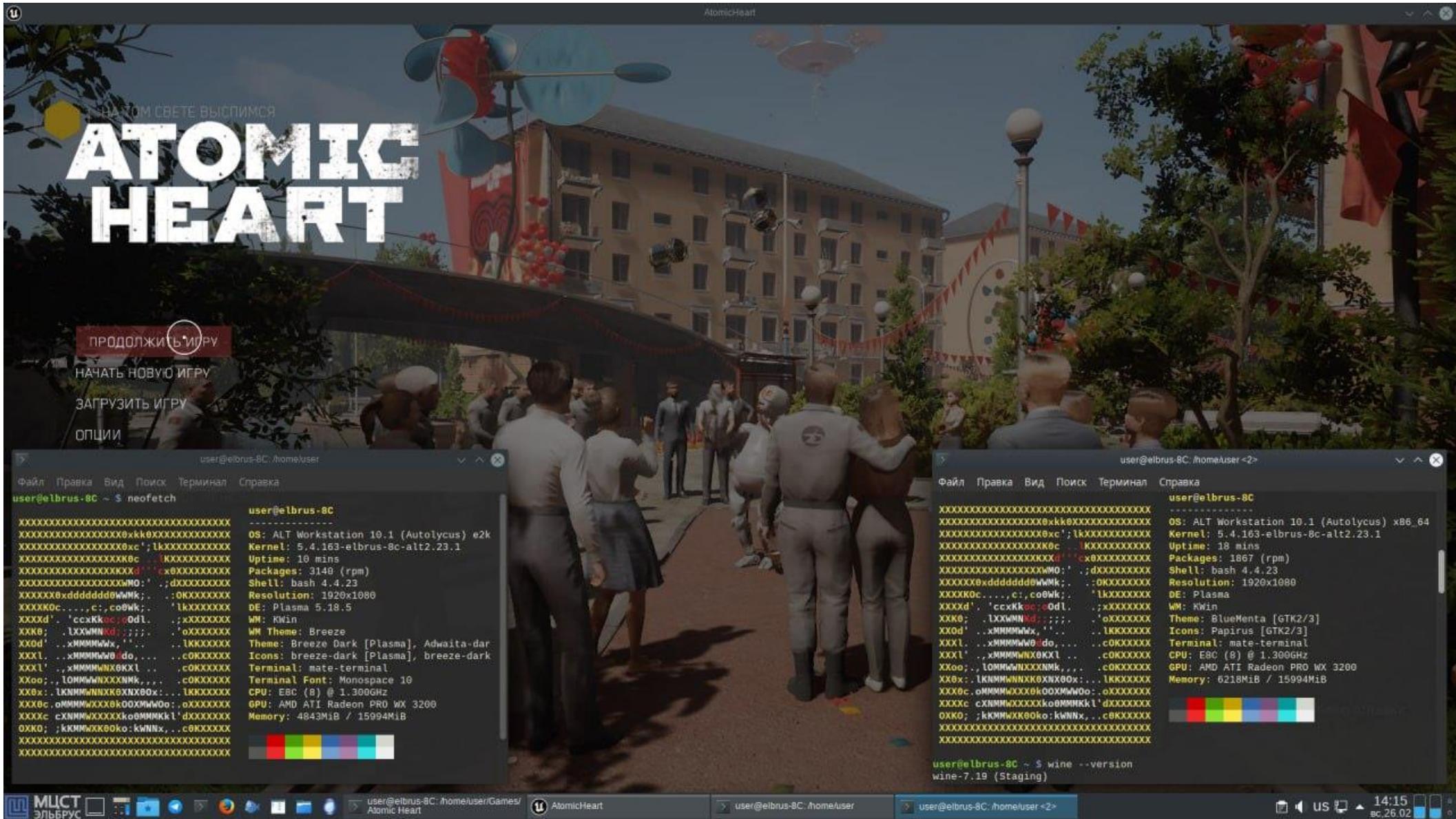


ELBRUS PC TEST

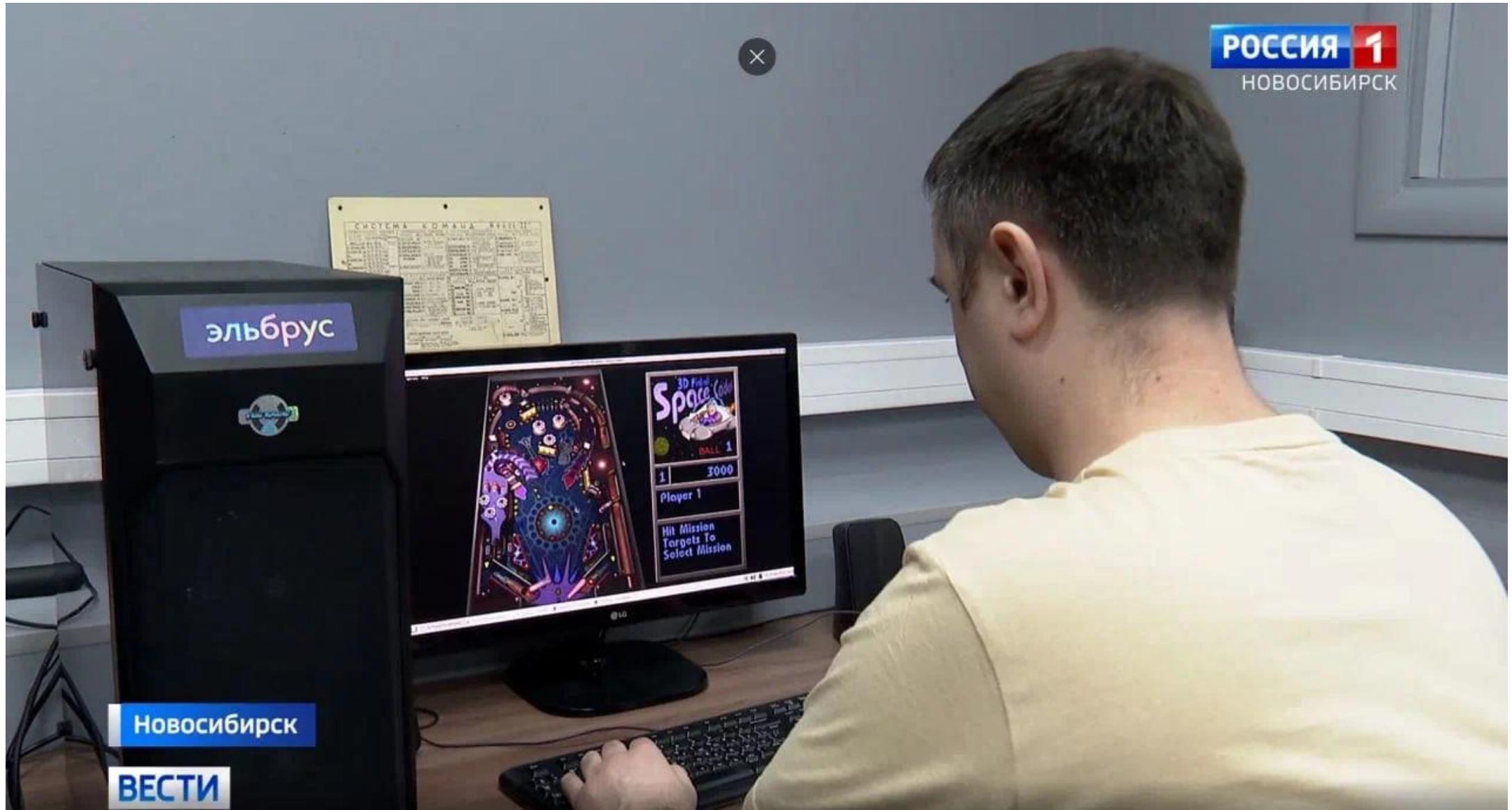


@ELBRUS_PC_TEST

Бенчмарки Эльбруса. Игры.



Бенчмарки Эльбруса. Игры. Дмитрий Бачило.



Бенчмарки Эльбруса. Игры.



Ссылки.



Чаты:

- Главный чат (https://t.me/e2k_chat),
- Флудилка Эльбруса (<https://t.me/+TasiREgUYeg4xRfa>),
- МЦСТ (<https://t.me/ElbrusCPUTeam>),
- Эмулятор qemu_e2k (https://t.me/qemu_e2k)



Каналы:

- Фан клуб (https://t.me/e2k_fans),
- Игры Эльбруса (https://t.me/elbrus_pc_test)
- Стикеры: Elbrus2000 (<https://telegram.me/addstickers/Elbrus2000>)



Общая информация:

- ALT Linux FAQ по Эльбрусу (<https://www.altlinux.org/%D1%8D%D0%BB%D1%8C%D0%B1%D1%80%D1%83%D1%81/faq>)
- Документация по Lintel (<https://storage.mcst.ru/index.php/s/1eeOA2R3CvRme19>)
- RTC (<https://storage.mcst.ru/index.php/s/WcykPhCnQtQkxC1>)



Разработка под e2k:

- Компилятор от x86 платформы с iso-диска (http://mcst.ru/elbrus_os) эльбруса может генерировать e2k код
- Проект qemu-e2k (https://t.me/qemu_e2k) позволяет запускать консольные приложения
- Можно получить бесплатный доступ к серверу Эльбрус по ssh, запросив его в чате @e2k_chat, https://t.me/elbrus_gensokyo
- Интерактивный компилятор (https://t.me/e2k_chat/104934) в ассемблер Эльбруса



Информация от МЦСТ:

- Руководство по эффективному программированию на платформе «Эльбрус» (http://mcst.ru/elbrus_prog)
- Учебное пособие "Микропроцессоры и вычислительные комплексы семейства «Эльбрус» (https://t.me/e2k_chat/47720)"



в открытых проектах e2k:

- Портирование игр (<https://trello.com/b/6JmJa2dd/to-do-e2k>) под Эльбрус. Куратор: @r_a_sattarov
- ALT Linux Team (https://www.altlinux.org/ALT_Linux_Team)
- Эмулятор qemu-e2k (https://t.me/qemu_e2k)

Удалённый доступ.

Бот (позволяет узнать, работает ли сервер и добавлен ли пользователь) / Bot (used to check whether any server is online, or username is registered):
@ElbrusNemunoBot

Запрос доступа к серверам / Request access to servers: <https://elbrus.kurisa.ch/>

Discord-сервер (с оповещениями о создании пользователей) / Discord server (newly created user notifications also posted there): <https://discord.gg/v79jts573n>

Администратор проекта: @makise_homura

Спецификации серверов / Servers available:

Yukari: 4 × Elbrus-8C eight-core CPU at 1200 MHz (32 cores), 256 GB RAM, OS Elbrus

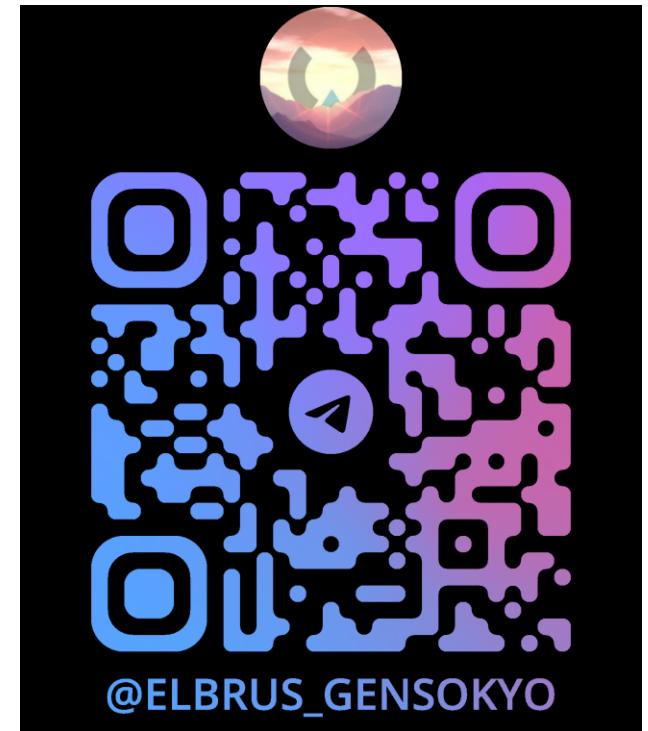
Mamizou: 3 × Elbrus-8C eight-core CPU at 1200 MHz (24 cores), 96 GB RAM, Alt Linux

Sumireko: 4 × Elbrus-8CV eight-core CPU at 1200 MHz (32 cores), 256 GB RAM, OS Elbrus

Raiko: 1 × Elbrus-2C+ dual-core CPU at 470 MHz (2 cores), 4 GB RAM, OS Elbrus

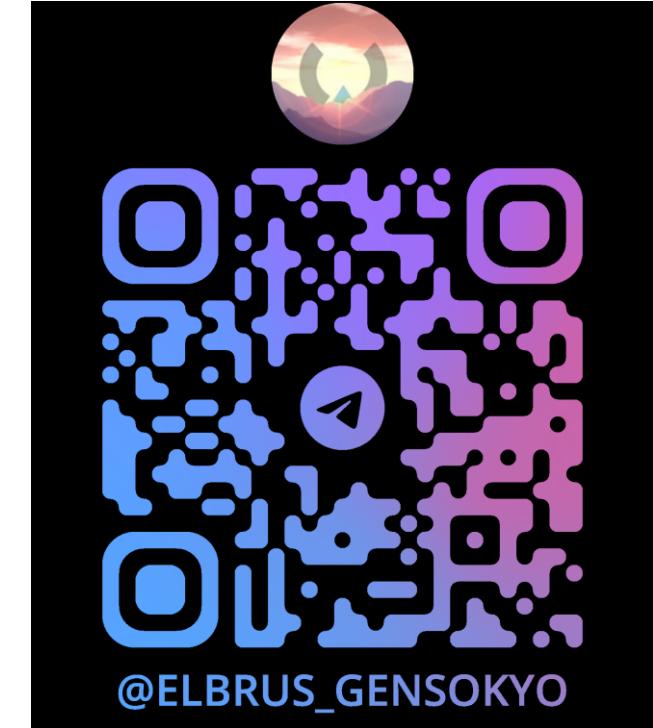
Yuyuko: 1 × Elbrus-8C eight-core CPU at 1200 MHz (8 cores), 32 GB RAM, Astra Linux

Shikieiki: 4 × Elbrus-4C quad-core CPU at 750 MHz (16 cores), 96 GB RAM, OS Elbrus



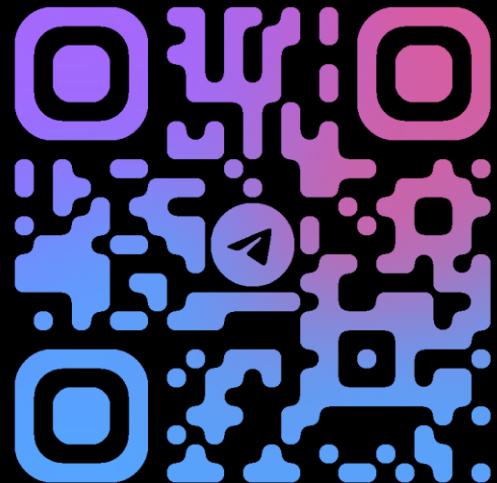
https://t.me/elbrus_gensokyo

Удалённый доступ.



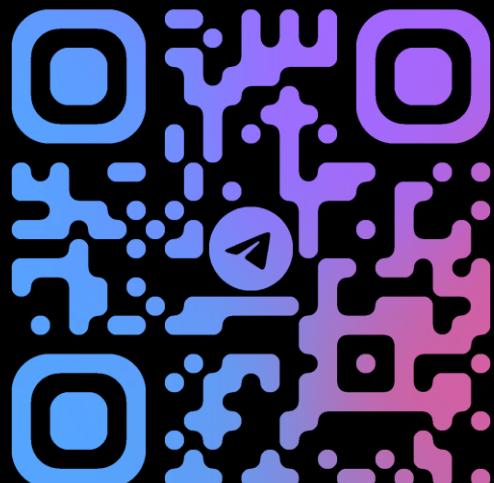
https://t.me/elbrus_gensokyo

Чаты и каналы



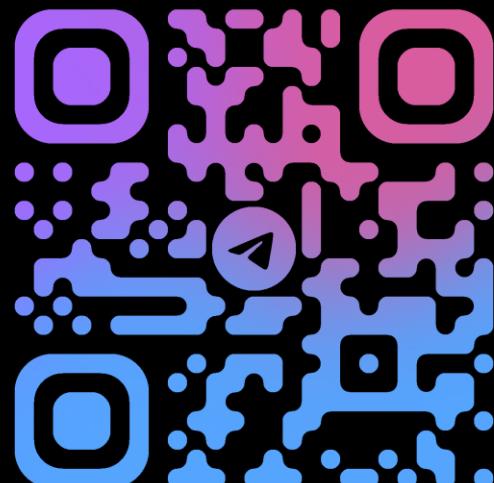
@E2K_FANS

https://t.me/e2k_fans



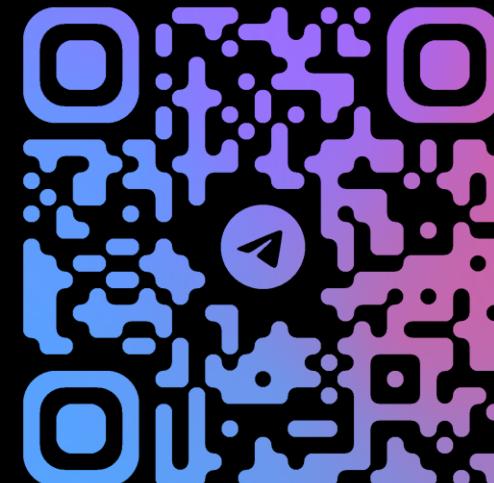
@elbrus

<https://t.me/elbrus>



@QEMU_E2K

https://t.me/qemu_e2k



@ELBRUS_GENOKYO

https://t.me/elbrus_gensokyo