Unleashing the Power of RISC-V E-Trace with a Highly Efficient Software Decoder

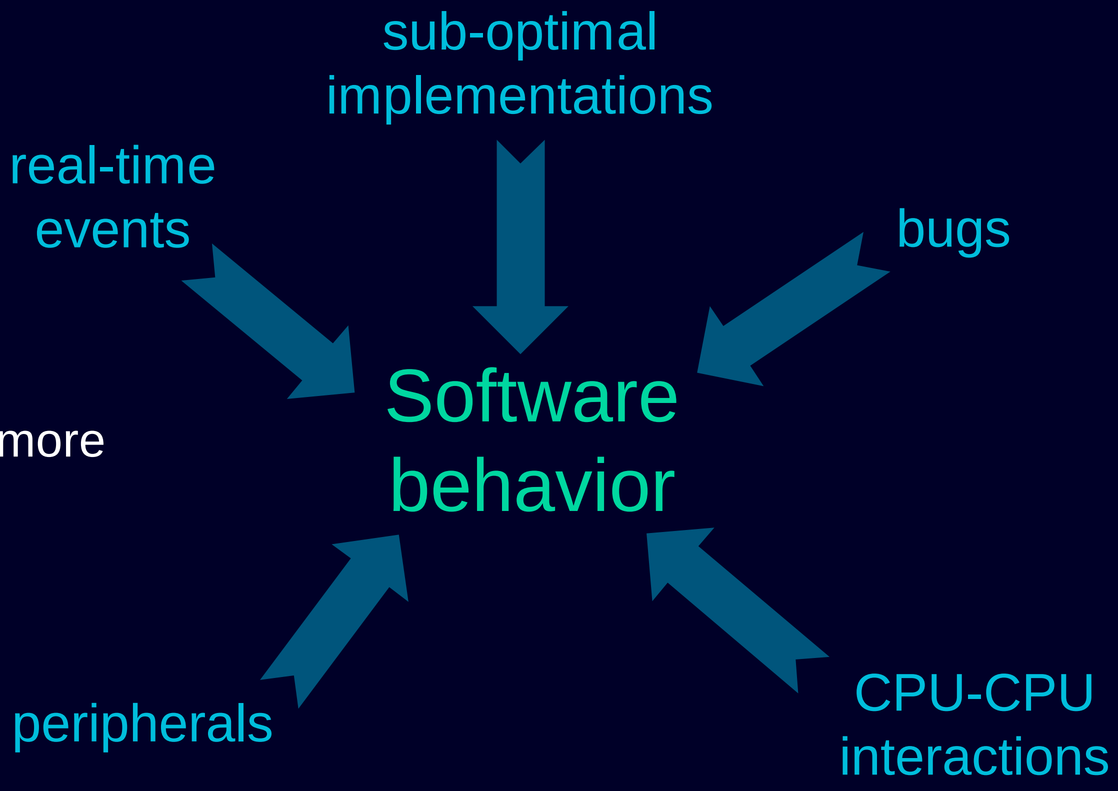
13 May 2025

Agenda

What is Trace?
Case Studies
Decoding Speed
Python API
Custom Instruction Decoding
Summary

What is Trace?

- Provides visibility for the processor behaviour
- Traditional techniques stops the processor
- Sampling adds operations and not full visibility
- In many applications traditional techniques don't work anymore
  - Time critical applications
  - Heisenbug bugs
  - Complex SoC with multiple cores
- Complement other debugging tools



sub-optimal implementations

real-time events

bugs

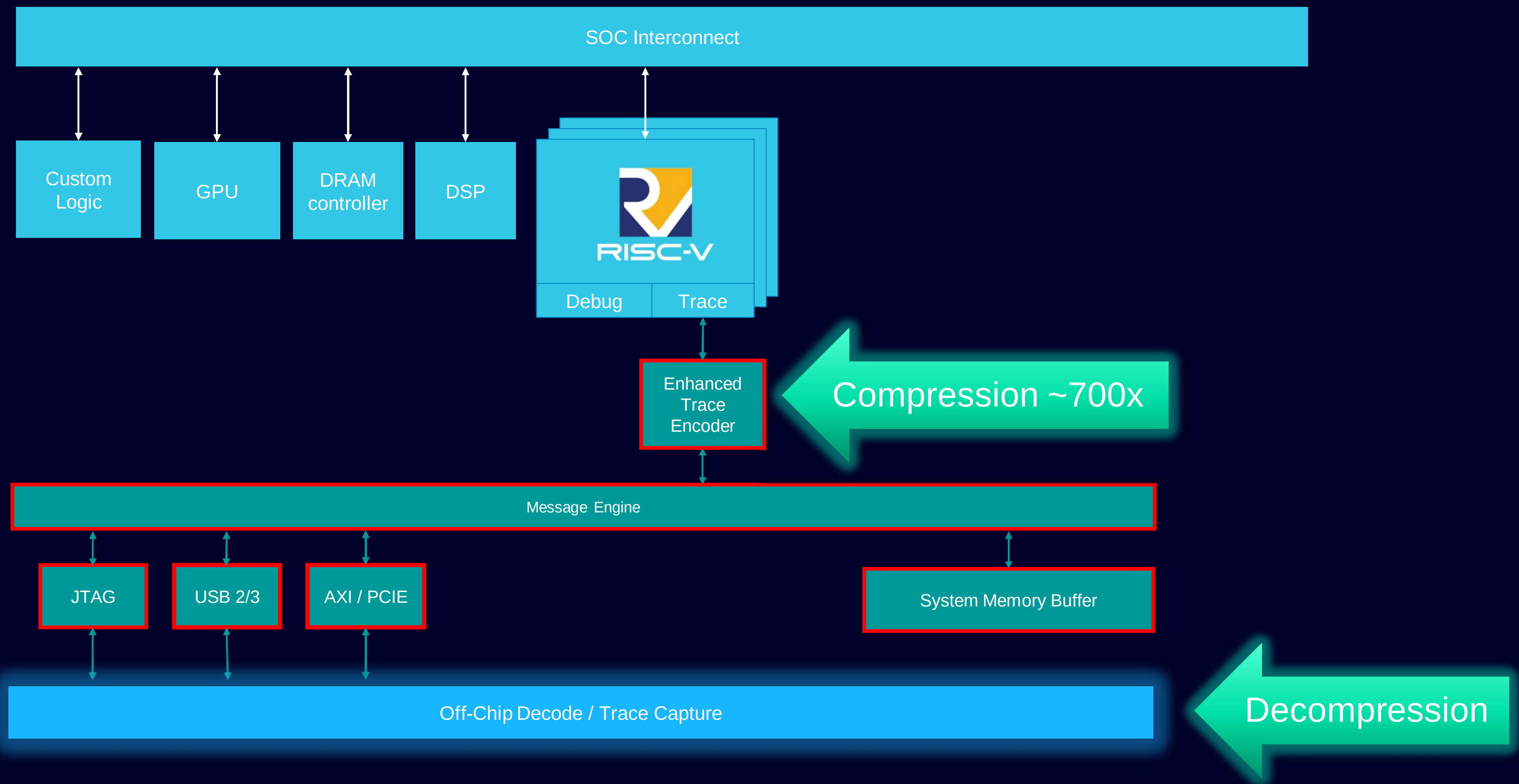
Software behavior

peripherals

CPU-CPU interactions

It is a technique for monitoring the activity of your processor in an unintrusive way.

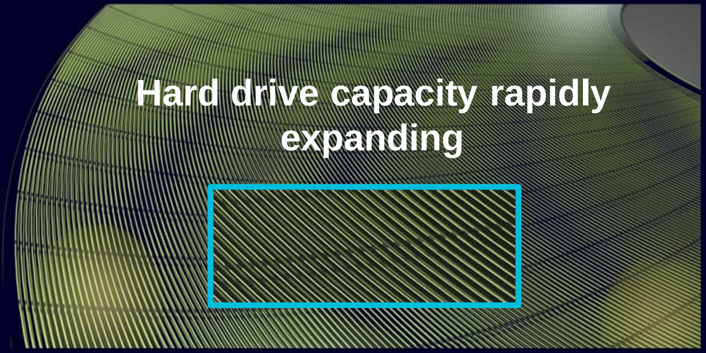
What is Trace?



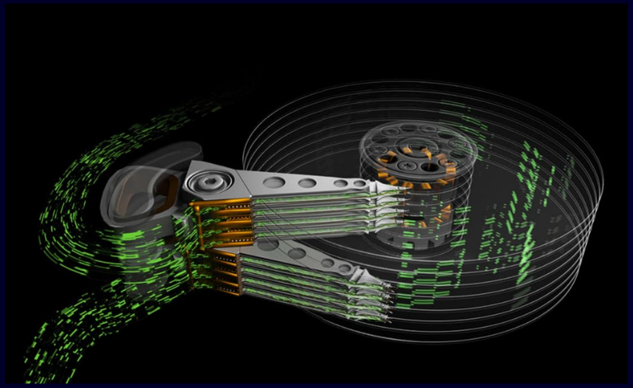
Case Study: Seagate Hard Drive SoC – Motion Control Application

Reference: Richard Bohn, “Debug & Optimization Strategy in Tomorrow’s Storage Technology” Seagate Technologies, Siemens U2U Presentation (available on Siemens EDA website)

The Problem



At 50TB, track density will exceed
1 million tracks per inch (TPI)
(2.4 nm positioning accuracy)



Multi-stage actuators for coarse movement and fine positioning

+

Real-Time Processing

- Disturbance detection algorithms
- Adaptive control features
- Feed-forward compensation
- High sample-rate computation

Constraints

- Power, space, and cost

SEAGATE

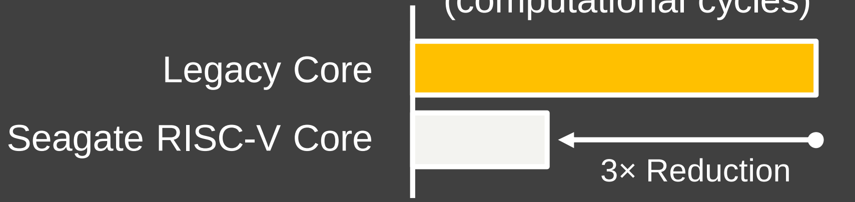
RISC-V

Microarchitecture optimization, parallelism, and latency reduction

RISC-V-Enabled Solution

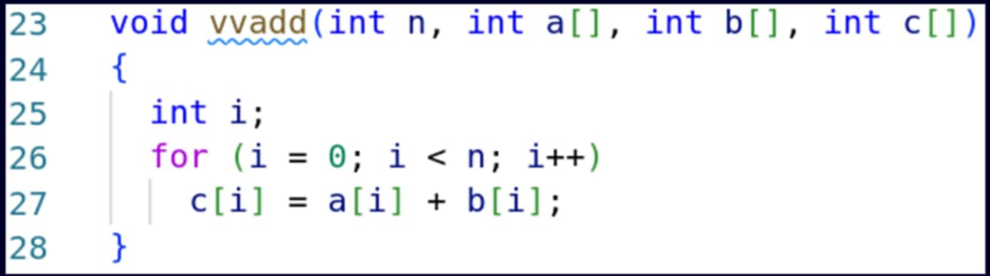
Functional Example:

Disturbance detection filter (computational cycles)

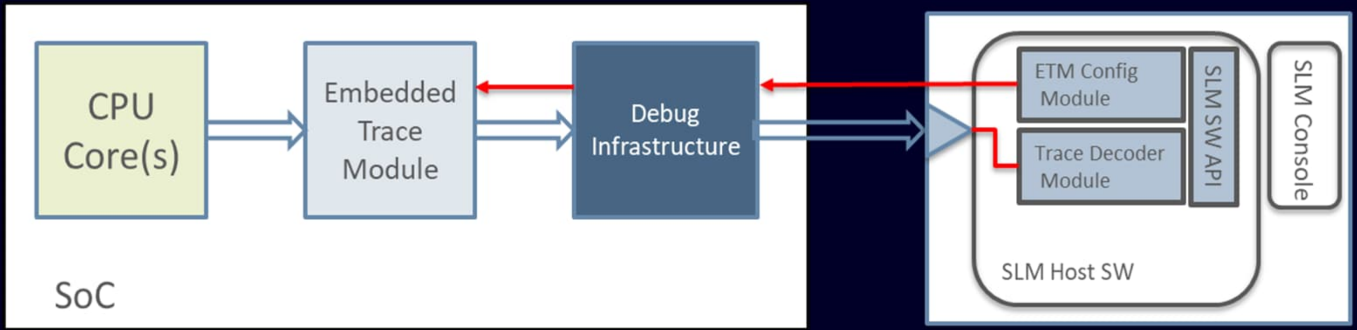


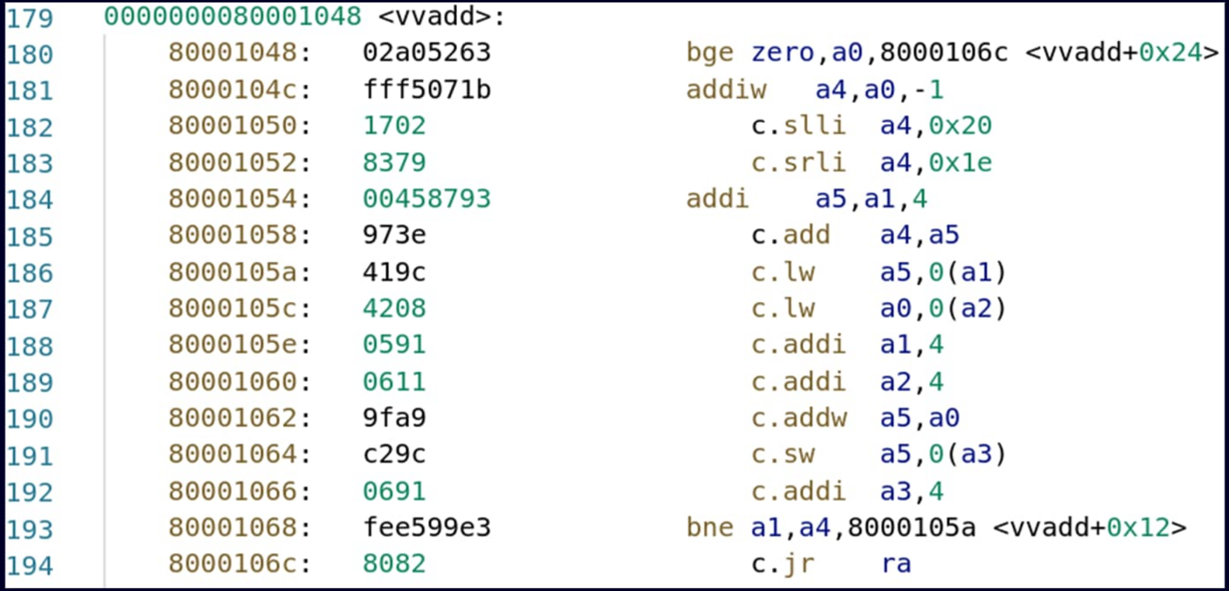
Integrating Trace Debug with SLM Applications

- Challenges of wide-scale deployment over silicon lifetime:
  - Latent manufacturing defects, voltage droop, excessive leakage, thermal, soft errors, aging, etc.
  - Manifest as software errors or under-performance
  - Need to monitor CPU instructions, bus transactions, and critical internal signals
  - This has to be aligned with data from silicon sensors, slack and voltage droop monitors, etc.
  - In-system (deterministic) test and Logic and Memory built-in self-test (BIST) response data can also be aligned with trace



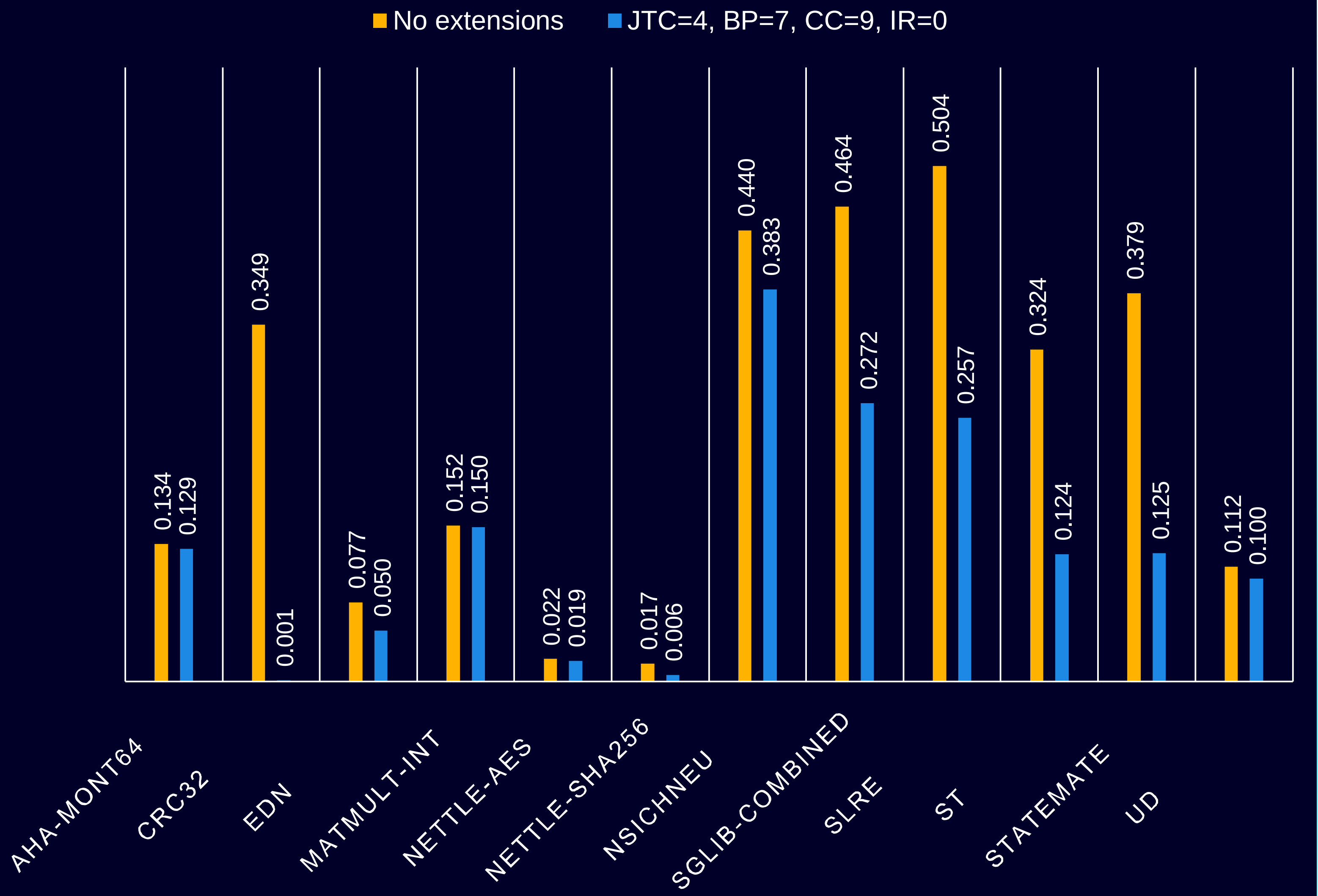
```c
void vvadd(int n, int a[], int b[], int c[])
{
    int i;
    for (i = 0; i < n; i++)
        c[i] = a[i] + b[i];
}
```



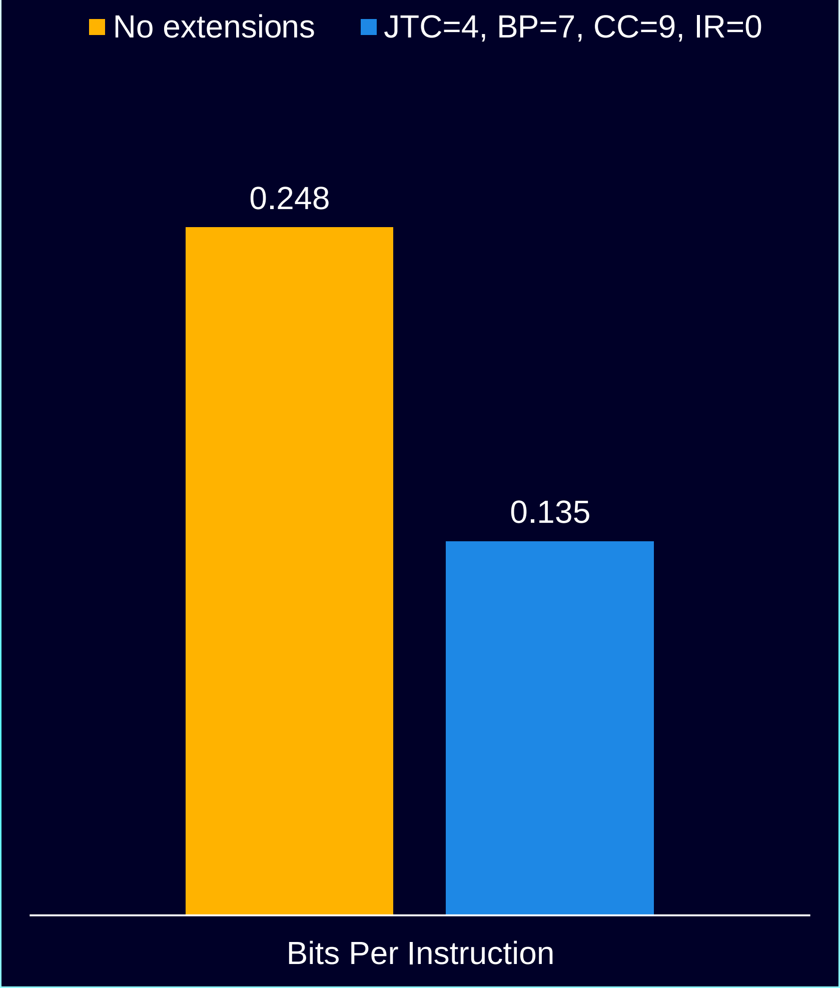


```
0000000080001048 <vvadd>:
    80001048:   02a05263        bge zero,a0,8000106c <vvadd+0x24>
    8000104c:   fff5071b        addiw   a4,a0,-1
    80001050:   1702                c.slli  a4,0x20
    80001052:   8379                c.srli  a4,0x1e
    80001054:   00458793        addi    a5,a1,4
    80001058:   973e                c.add   a4,a5
    8000105a:   419c                c.lw    a5,0(a1)
    8000105c:   4208                c.lw    a0,0(a2)
    8000105e:   0591                c.addi  a1,4
    80001060:   0611                c.addi  a2,4
    80001062:   9fa9                c.addw  a5,a0
    80001064:   c29c                c.sw    a5,0(a3)
    80001066:   0691                c.addi  a3,4
    80001068:   fee599e3        bne a1,a4,8000105a <vvadd+0x12>
    8000106c:   8082                c.jr    ra
```

Embench™ : HW Trace Encoder Compression – bits per instruction

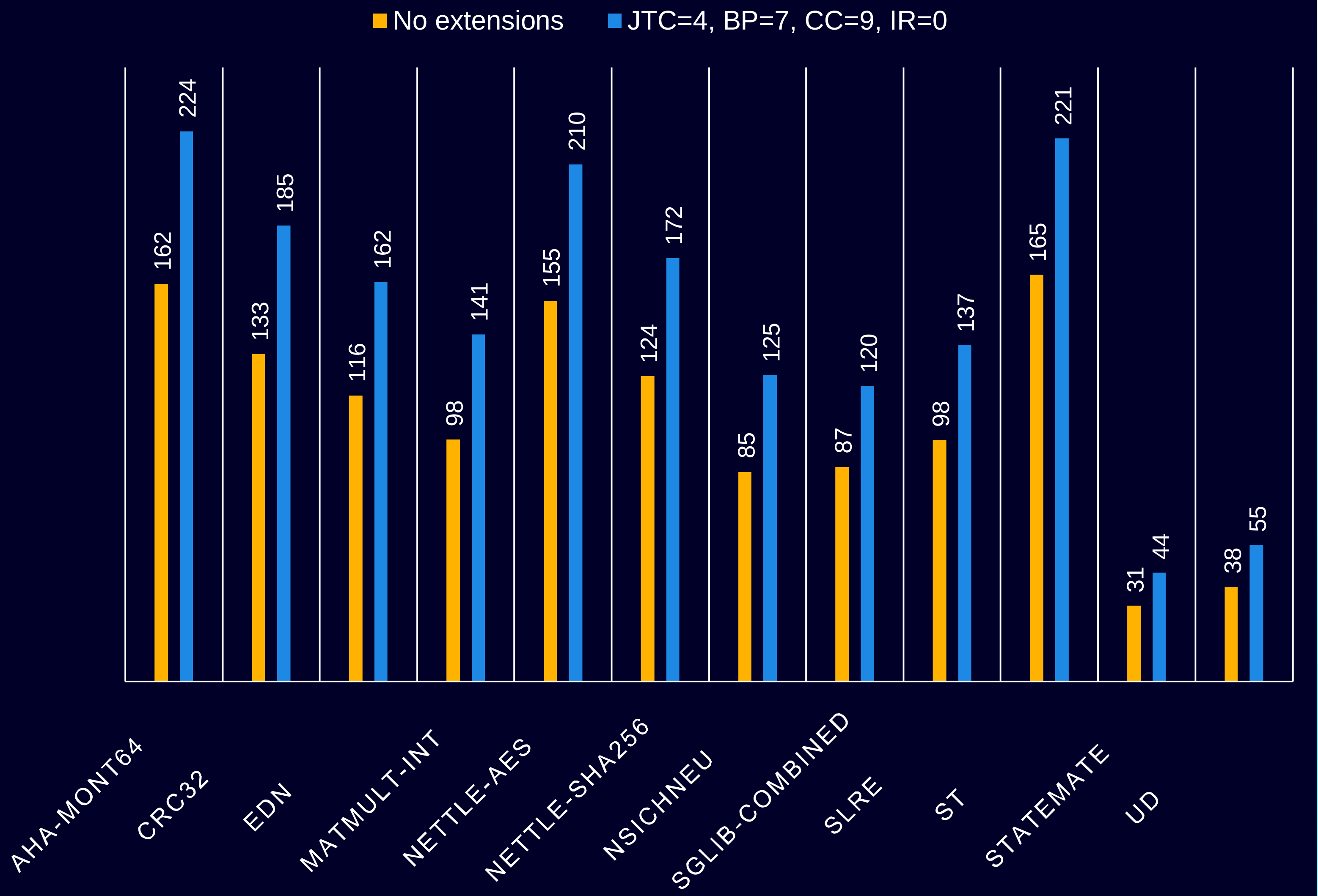


46% DECREASE

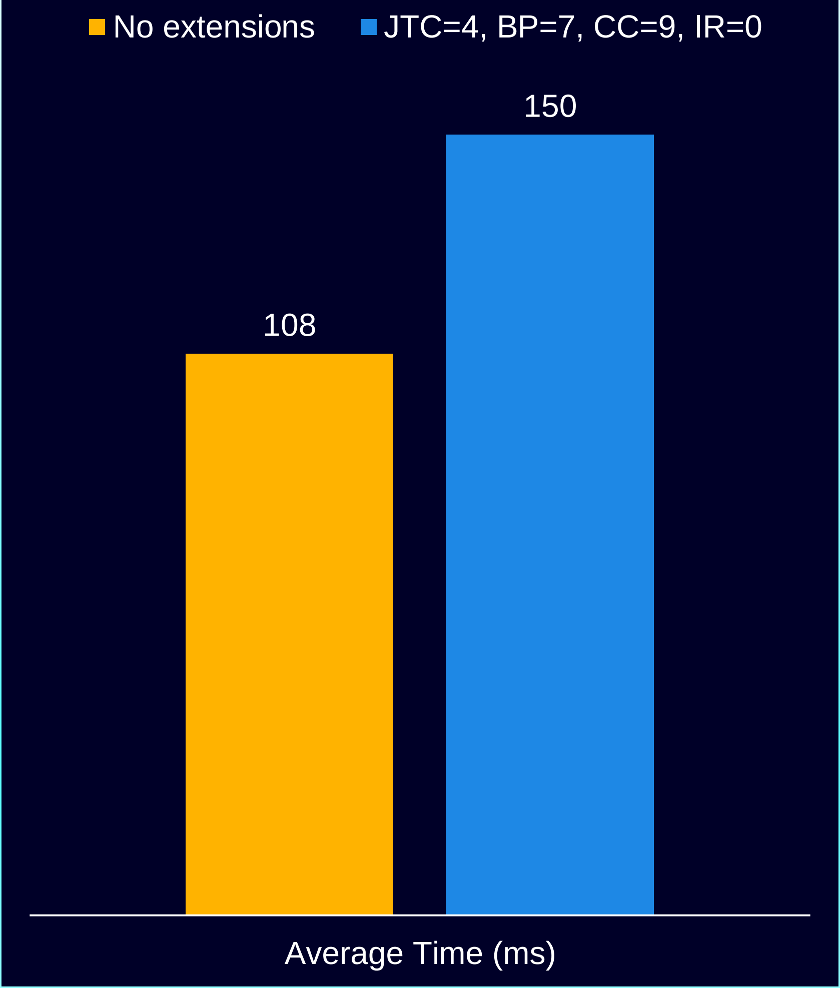


CPU: Intel® Xeon® W-2225 @ 4.10GHz; RAM: 2x8GB (HMA81GR7CJR8N-XN) @ 2934 MT/s; OS: Rocky Linux 8.10 (Green Obsidian)

Embench™ SW Trace Decoding Speed – time in milliseconds

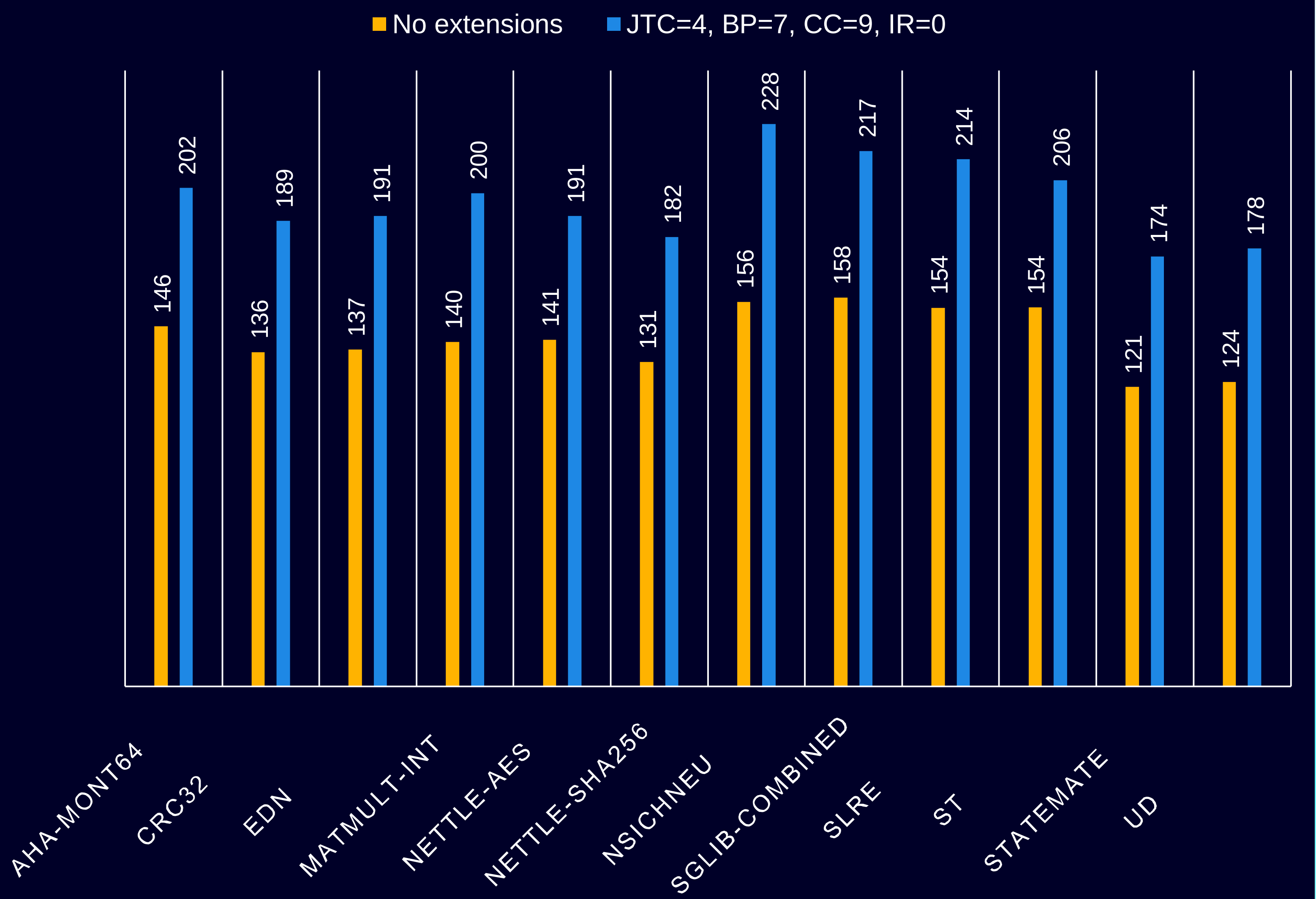


39% INCREASE

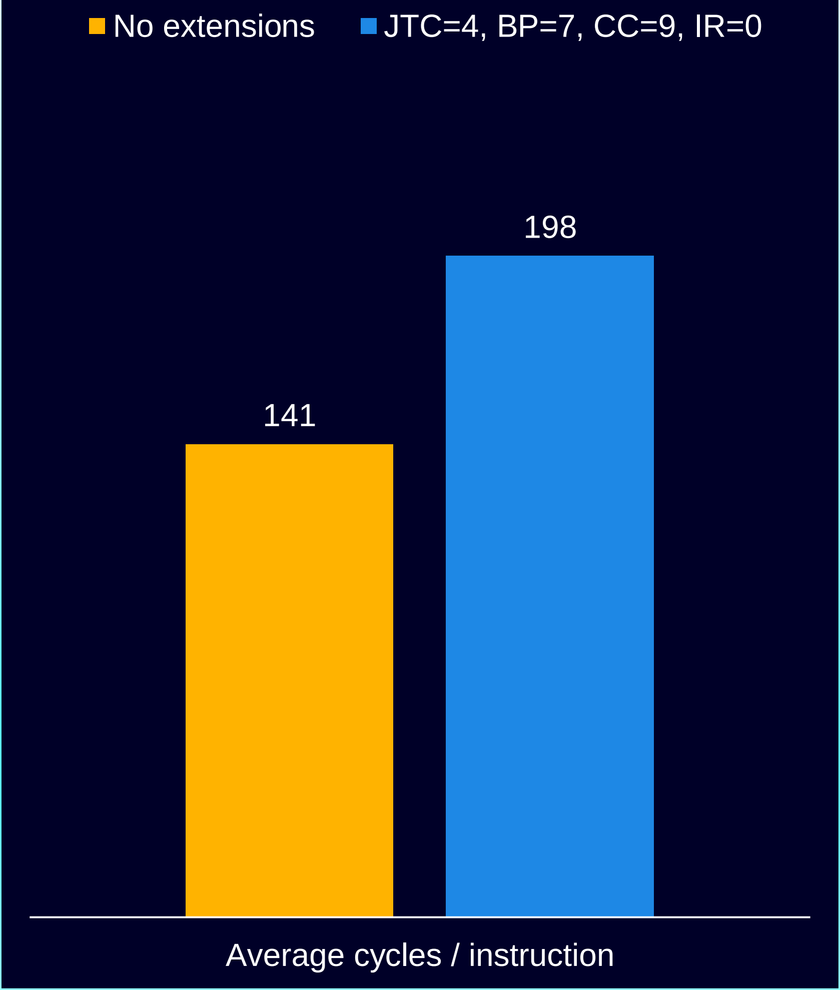


CPU: Intel® Xeon® W-2225 @ 4.10GHz; RAM: 2x8GB (HMA81GR7CJR8N-XN) @ 2934 MT/s; OS: Rocky Linux 8.10 (Green Obsidian)

Embench™ SW Trace Decoding Speed – cycles per instruction

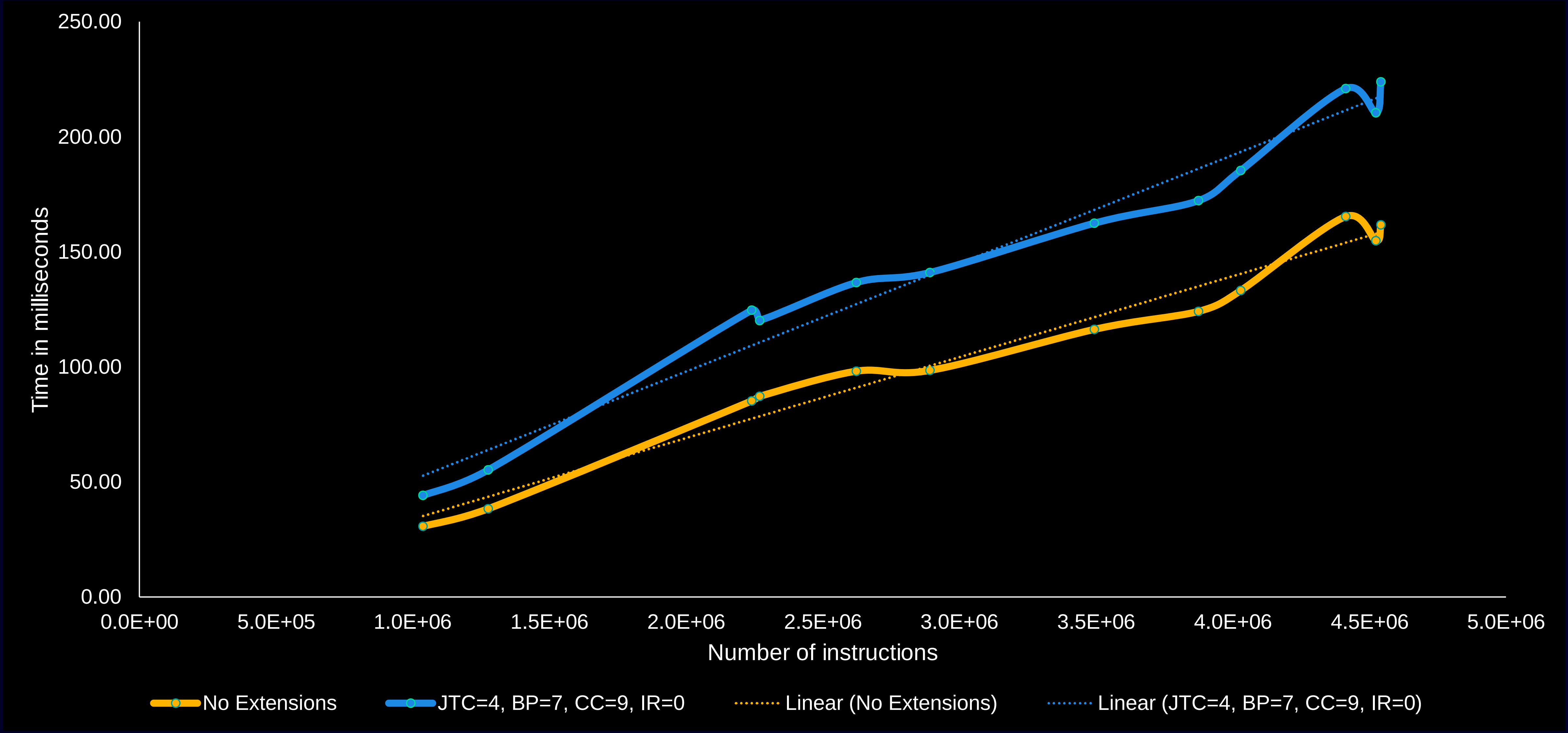


39% INCREASE



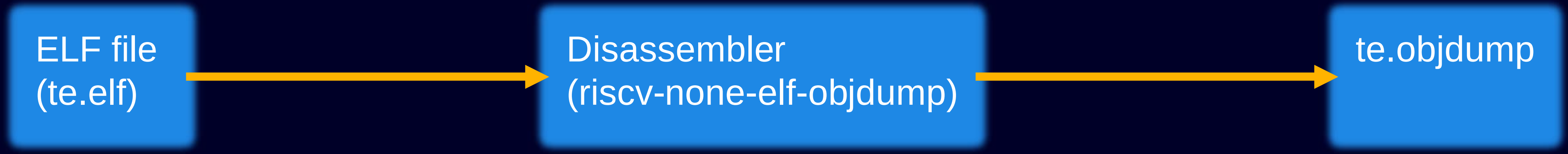
CPU: Intel® Xeon® W-2225 @ 4.10GHz; RAM: 2x8GB (HMA81GR7CJR8N-XN) @ 2934 MT/s; OS: Rocky Linux 8.10 (Green Obsidian); Cycles measured with __rdtsc **function from** <intrin.h>

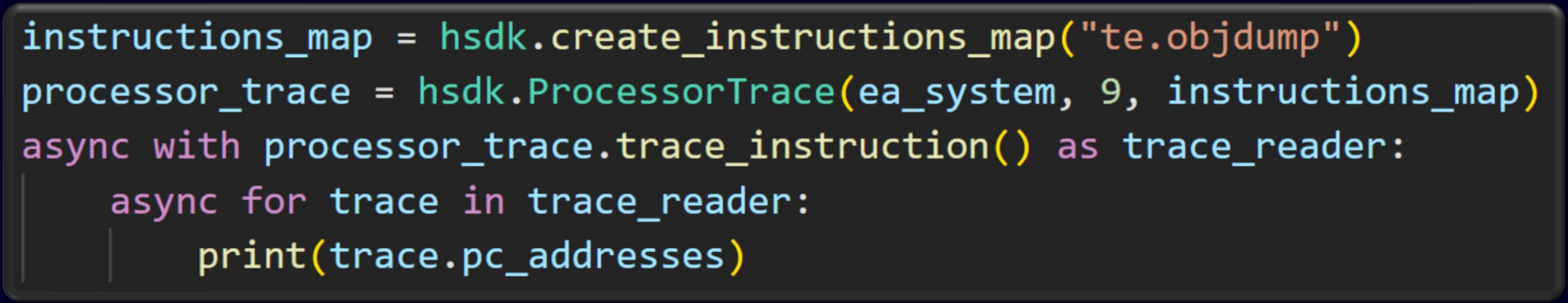
Embench™ – Empirical SW Trace Decoding Big-O complexity - O(n)



CPU: Intel® Xeon® W-2225 @ 4.10GHz; RAM: 2x8GB (HMA81GR7CJR8N-XN) @ 2934 MT/s; OS: Rocky Linux 8.10 (Green Obsidian)

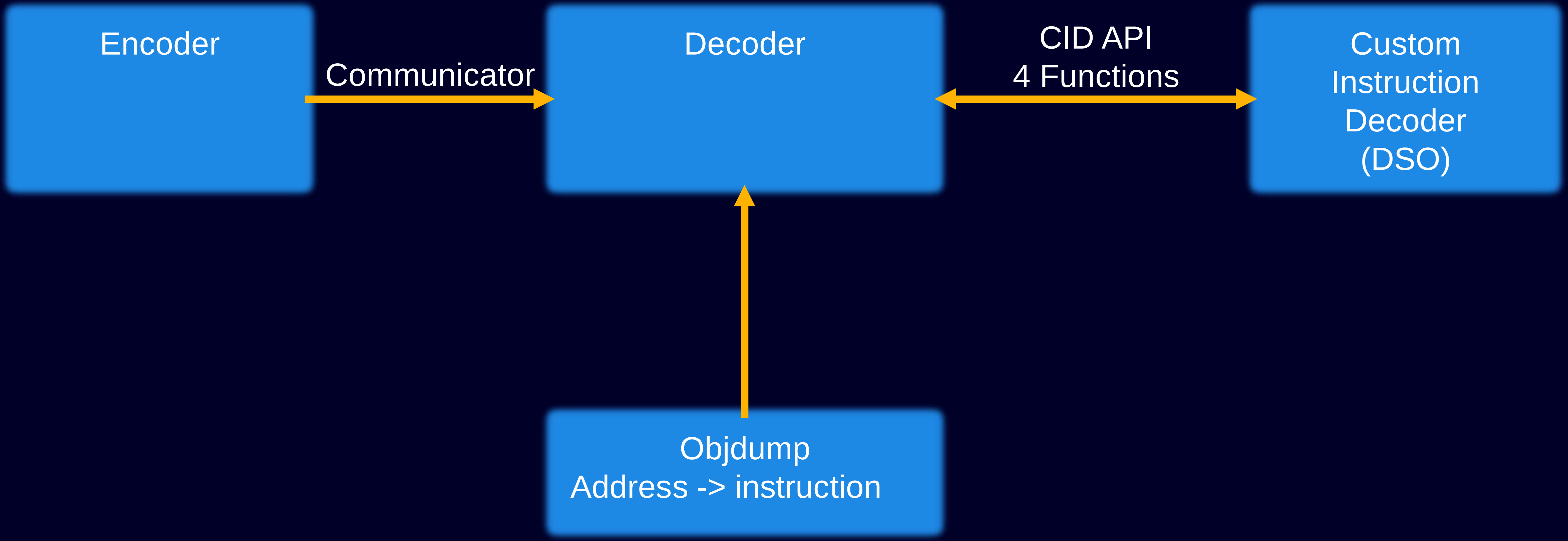
Python API





```python
instructions_map = hsdk.create_instructions_map("te.objdump")
processor_trace = hsdk.ProcessorTrace(ea_system, 9, instructions_map)
async with processor_trace.trace_instruction() as trace_reader:
    async for trace in trace_reader:
        print(trace.pc_addresses)
```

Custom Instruction Decoding (CID)



Encoder

Communicator

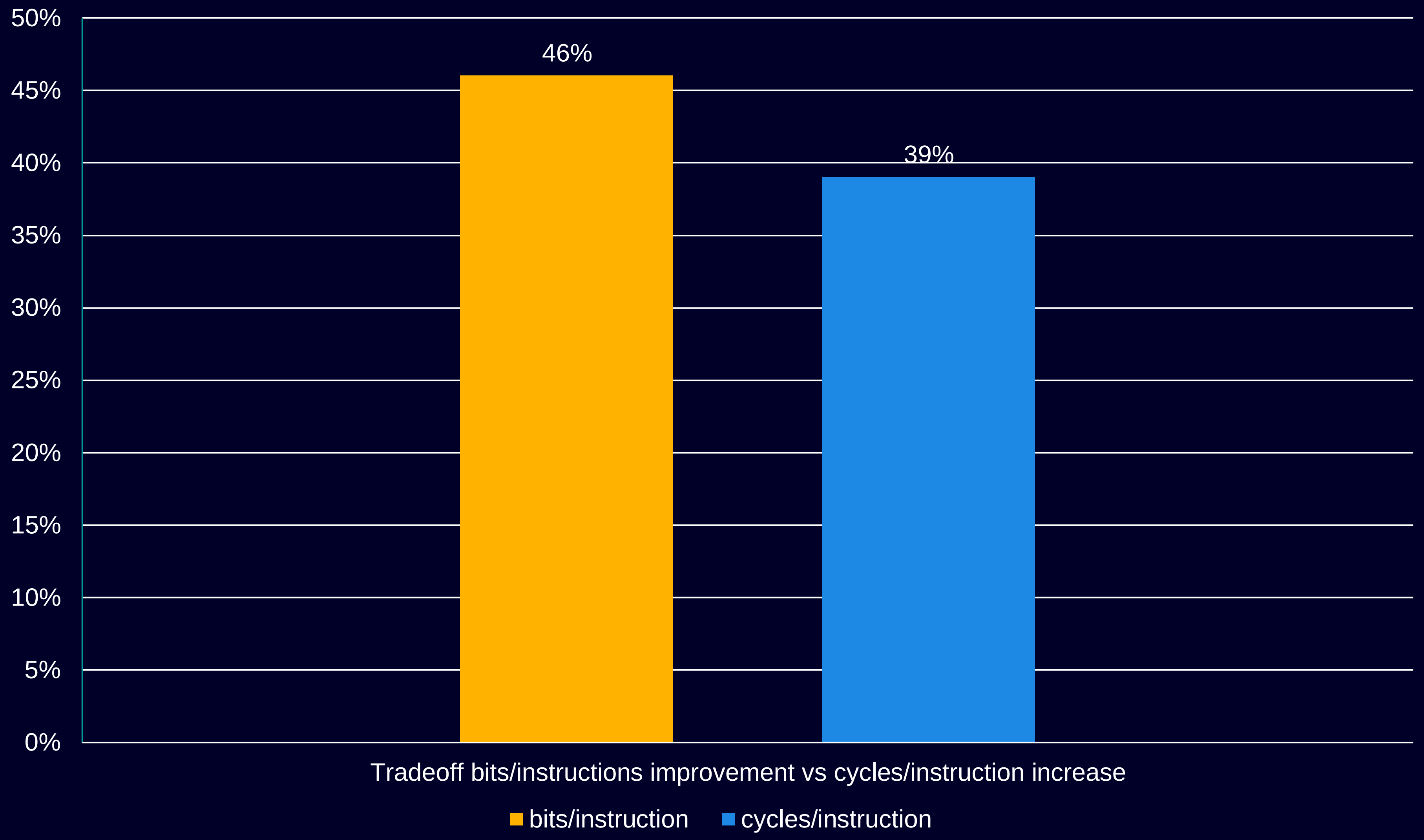
Decoder

CID API
4 Functions

Custom
Instruction
Decoder
(DSO)

Objdump
Address -> instruction

Benefits and Tradeoffs of E-Trace Optional Extensions



Summary & Conclusion

- Implemented an efficient RISC-V Trace Decoder in software with C++ API and Python access to build Debug and Silicon Lifecycle Management applications
- Results on Embench™ Benchmark programs yield a SW Trace Decoder average execution time of 108 ms with default setting
- Optional Extensions- Decoding time increases by 39% on average with benefit of 46% improvement in trace compression
- The benefits of having a complete solution for E-Trace HW Encoding and SW Decoding are shown with two RISC-V based case studies- a storage controller with a 2.4 nm positioning accuracy and a Silicon Lifecycle Monitoring and debug application

Contact

Unleashing the Power of RISC-V E-Trace with a Highly Efficient Software Decoder

Marcel Zak

Software Engineer

Tessent Embedded Analytics
Siemens Digital Industries Software
20 Station Road
Cambridge, CB1 2JY
United Kingdom

E-mail marcel.zak@siemens.com