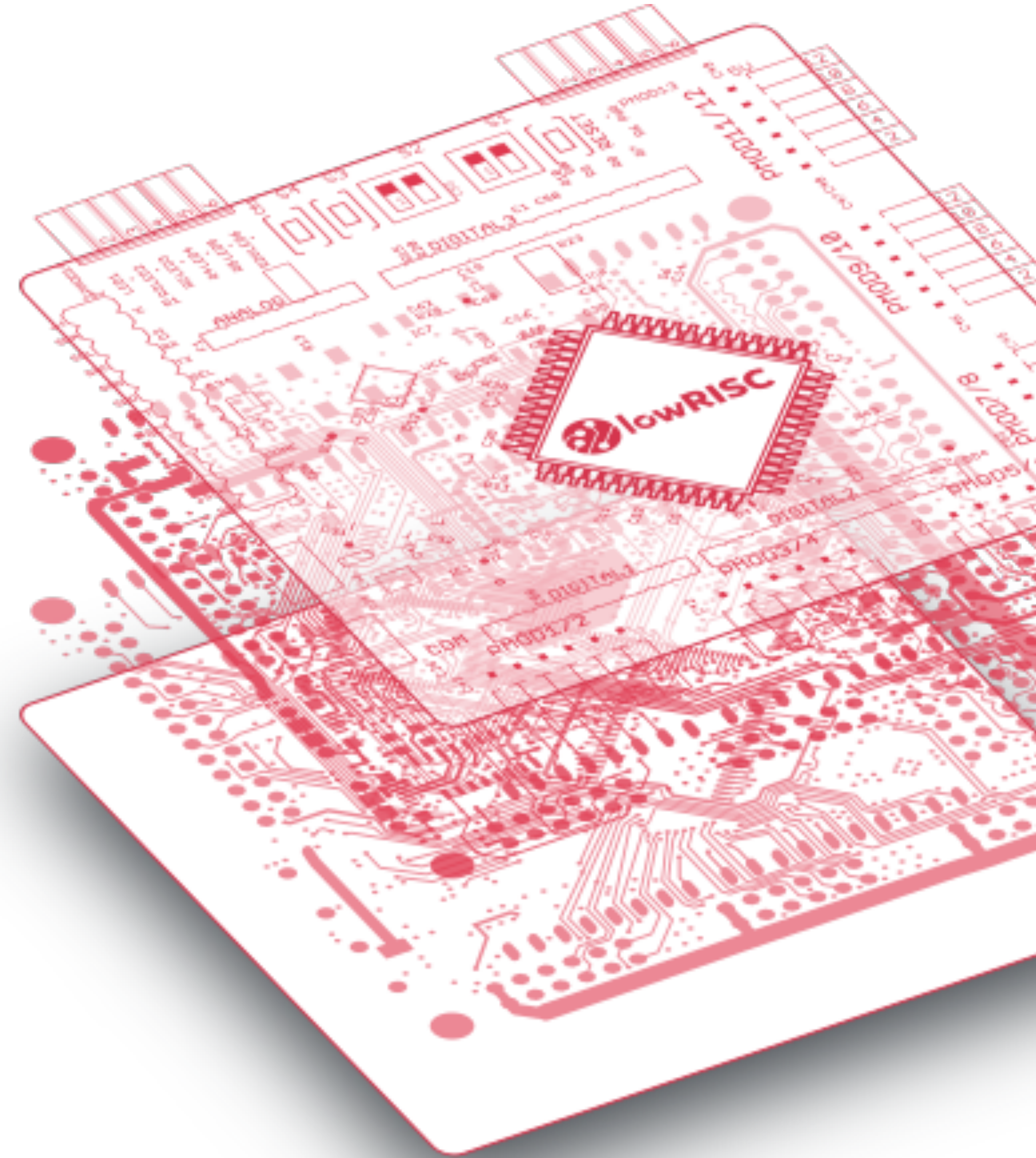


# The RISC-V psABI





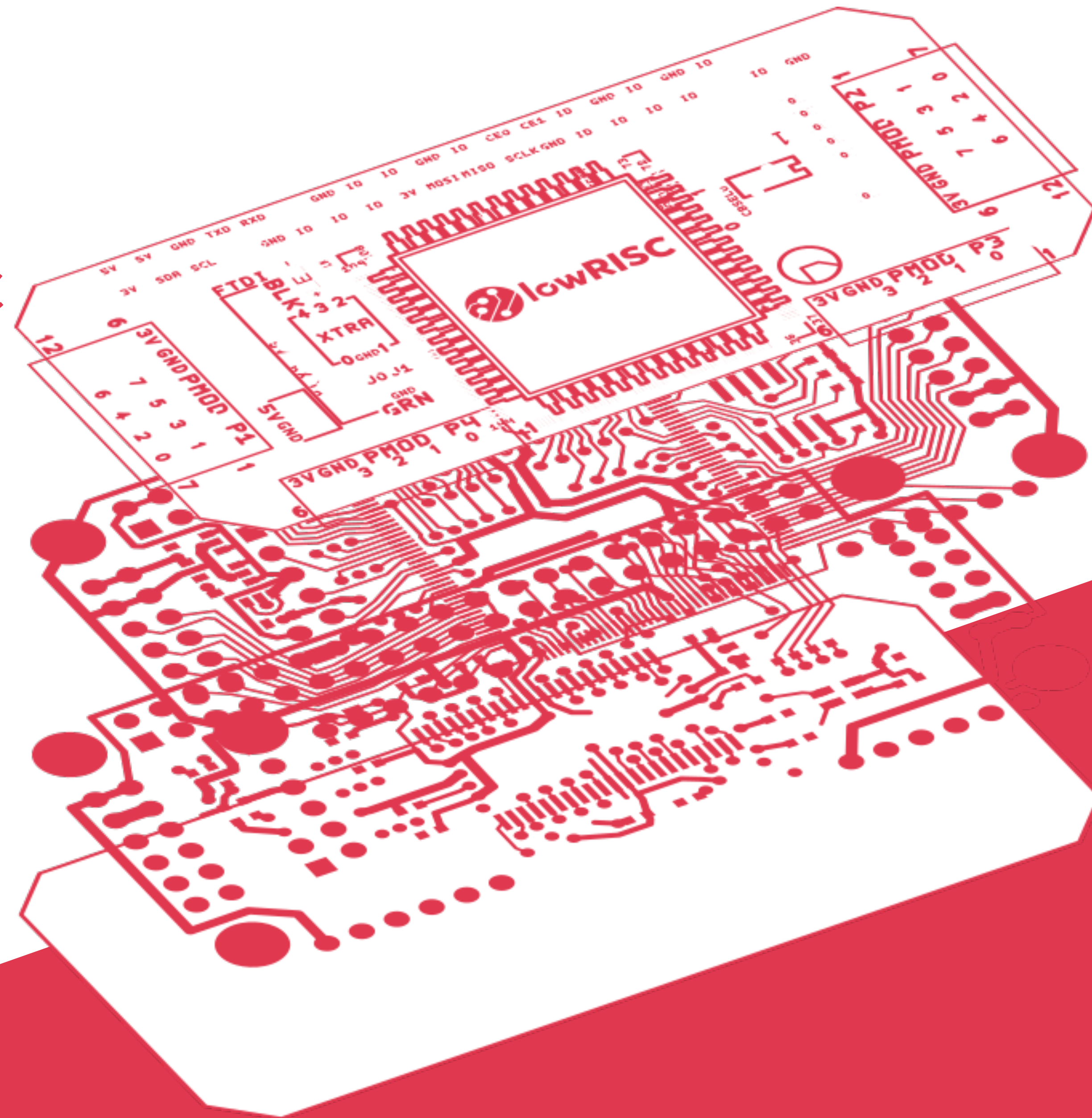
# Who am I?

Senior Software Engineer and  
Software Team Lead at lowRISC

Former UW PhD Student

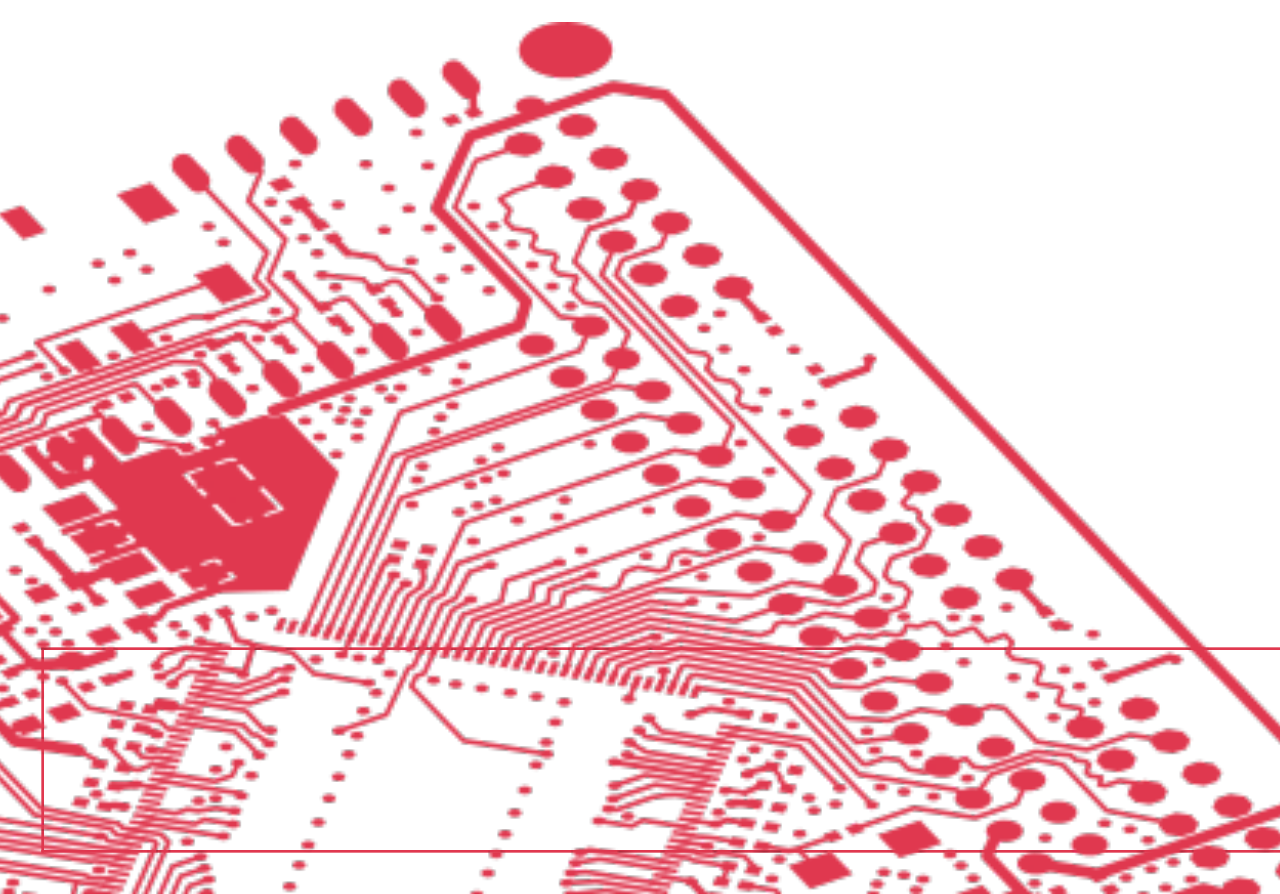
Contributor to:

- LLVM's RISC-V Backend
- RISC-V psABI





# Who are lowRISC?



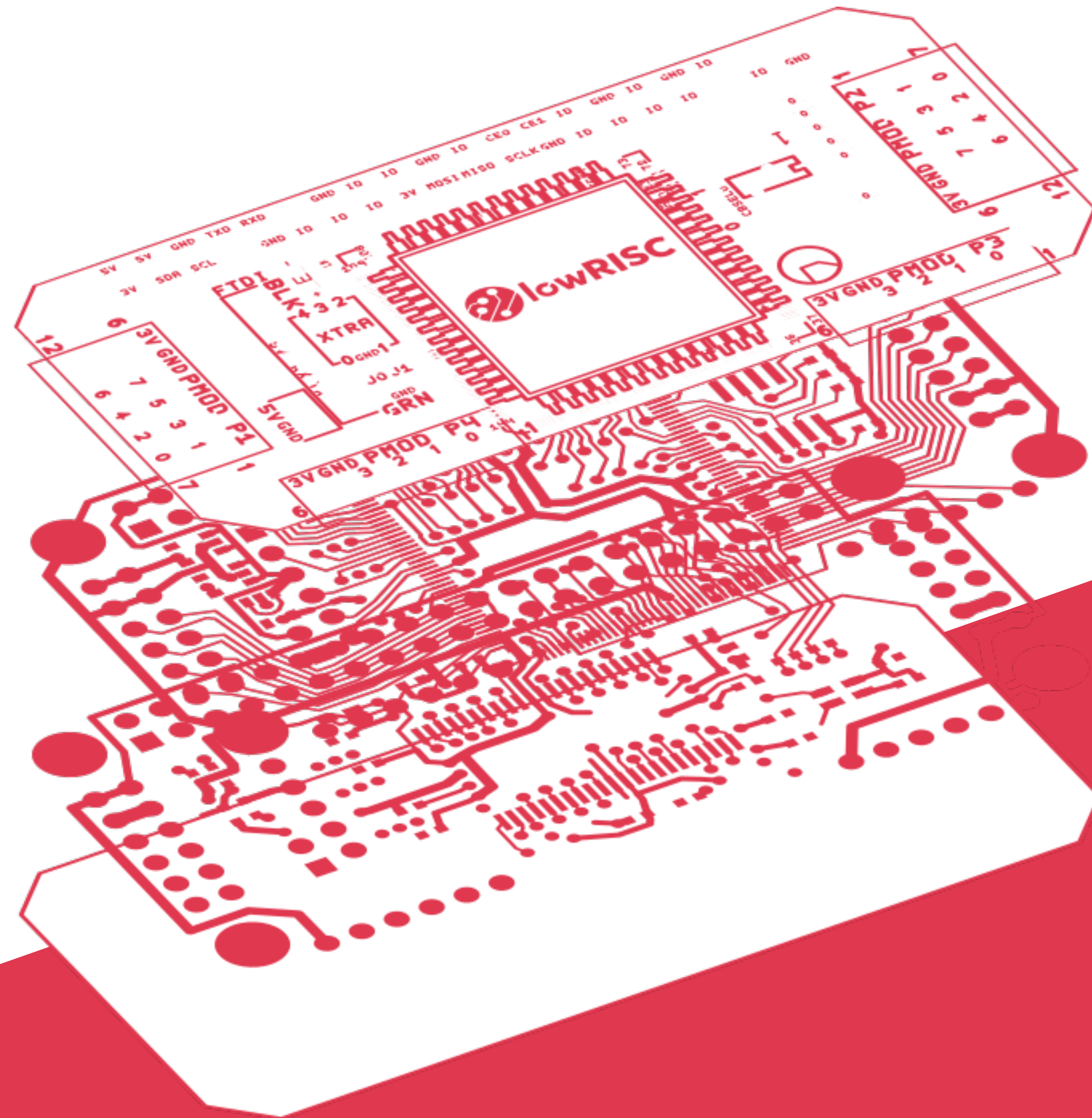


# opentitan



# This Lecture

- What is an ABI?
- The RISC-V psABI
- Embedded Systems
- Running Programs
- A New Embedded ABI?





# Let's talk APIs

Python » English » 3.8.3 » Documentation » The Python Standard Library »

Quick search

Go

previous | next | modules | index

Previous topic  
Introduction

Next topic  
Built-in Constants

This Page  
Report a Bug  
Show Source

## Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

		Built-in Functions		
<code>abs()</code>	<code>delattr()</code>	<code>hash()</code>	<code>memoryview()</code>	<code>set()</code>
<code>all()</code>	<code>dict()</code>	<code>help()</code>	<code>min()</code>	<code>setattr()</code>
<code>any()</code>	<code>dir()</code>	<code>hex()</code>	<code>next()</code>	<code>slice()</code>
<code>ascii()</code>	<code>divmod()</code>	<code>id()</code>	<code>object()</code>	<code>sorted()</code>
<code>bin()</code>	<code>enumerate()</code>	<code>input()</code>	<code>oct()</code>	<code>staticmethod()</code>
<code>bool()</code>	<code>eval()</code>	<code>int()</code>	<code>open()</code>	<code>str()</code>
<code>breakpoint()</code>	<code>exec()</code>	<code>isinstance()</code>	<code>ord()</code>	<code>sum()</code>
<code>bytearray()</code>	<code>filter()</code>	<code>issubclass()</code>	<code>pow()</code>	<code>super()</code>
<code>bytes()</code>	<code>float()</code>	<code>iter()</code>	<code>print()</code>	<code>tuple()</code>
<code>callable()</code>	<code>format()</code>	<code>len()</code>	<code>property()</code>	<code>type()</code>
<code>chr()</code>	<code>frozenset()</code>	<code>list()</code>	<code>range()</code>	<code>vars()</code>
<code>classmethod()</code>	<code>getattr()</code>	<code>locals()</code>	<code>repr()</code>	<code>zip()</code>
<code>compile()</code>	<code>globals()</code>	<code>map()</code>	<code>reversed()</code>	<code>__import__()</code>
<code>complex()</code>	<code>hasattr()</code>	<code>max()</code>	<code>round()</code>	

**abs(x)**  
Return the absolute value of a number. The argument may be an integer or a floating point number. If the argument is a complex number, its magnitude is returned. If `x` defines `__abs__()`, `abs(x)` returns `x.__abs__()`.

**all(iterable)**  
Return `True` if all elements of the *iterable* are true (or if the iterable is empty). Equivalent to:

```
def all(iterable):
    for element in iterable:
        if not element:
            return False
    return True
```

**any(iterable)**  
Return `True` if any element of the *iterable* is true. If the iterable is empty, return `False`. Equivalent to:

```
def any(iterable):
    for element in iterable:
        if element:
            return True
    return False
```

**ascii(object)**  
As `repr()`, return a string containing a printable representation of an object, but escape the non-ASCII characters in the string returned by `repr()` using `\x`, `\u` or `\U` escapes. This generates a string similar to that returned by `repr()` in Python 2.

**bin(x)**  
Convert an integer number to a binary string prefixed with "0b". The result is a valid Python expression. If `x` is not a Python `int` object, it has to define an `__index__()` method that returns an integer. Some examples:

```
>>> bin(3)
'0b11'
>>>
```

GitHub Developer

Docs » Blog Forum Versions »

Search...

REST API v3

Reference Guides Libraries

## Overview

This describes the resources that make up the official GitHub REST API v3. If you have any problems or requests, please contact [GitHub Support](#) or [GitHub Premium Support](#).

- Current version
- Schema
- Authentication
- Parameters
- Root endpoint
- GraphQL global node IDs
- Client errors
- HTTP redirects
- HTTP verbs
- Hypermedia
- Pagination
- Rate limiting
- User agent required
- Conditional requests
- Cross origin resource sharing
- JSON-P callbacks
- Timezones

### Current version

By default, all requests to `https://api.github.com` receive the **v3** version of the REST API. We encourage you to [explicitly request this version](#) via the `Accept` header.

```
Accept: application/vnd.github.v3+json
```

For information about GitHub's GraphQL API v4, see the [v4 documentation](#). For information about migrating to GraphQL, see ["Migrating from REST."](#)

### Schema

All API access is over HTTPS, and accessed from `https://api.github.com`. All data is sent and received as JSON.

```
curl -i https://api.github.com/users/octocat/orgs
HTTP/1.1 200 OK
Server: nginx
Date: Fri, 12 Oct 2012 23:33:14 GMT
Content-Type: application/json; charset=utf-8
Connection: keep-alive
Status: 200 OK
ETag: "a00049ba79152d03308c34652f2cb612"
X-GitHub-Media-Type: github.v3
X-RateLimit-Limit: 5000
X-RateLimit-Remaining: 4987
X-RateLimit-Reset: 1350005394
Content-Length: 5
Cache-Control: max-age=0, private, must-revalidate
X-Content-Type-Options: nosniff
```

Blank fields are included as `null` instead of being omitted.

All timestamps return in ISO 8601 format:

```
2012-10-12T23:33:14Z
```

Overview

Media Types

OAuth Authorizations API

Other Authentication Methods

Troubleshooting

API Previews

Versions

Activity

Checks

Code Scanning

Gists

Git Data

GitHub Actions

GitHub Apps

GitHub Marketplace

Interactions

Issues

Migrations

Miscellaneous

Organizations

Projects

Pull Requests

Reactions

Repositories

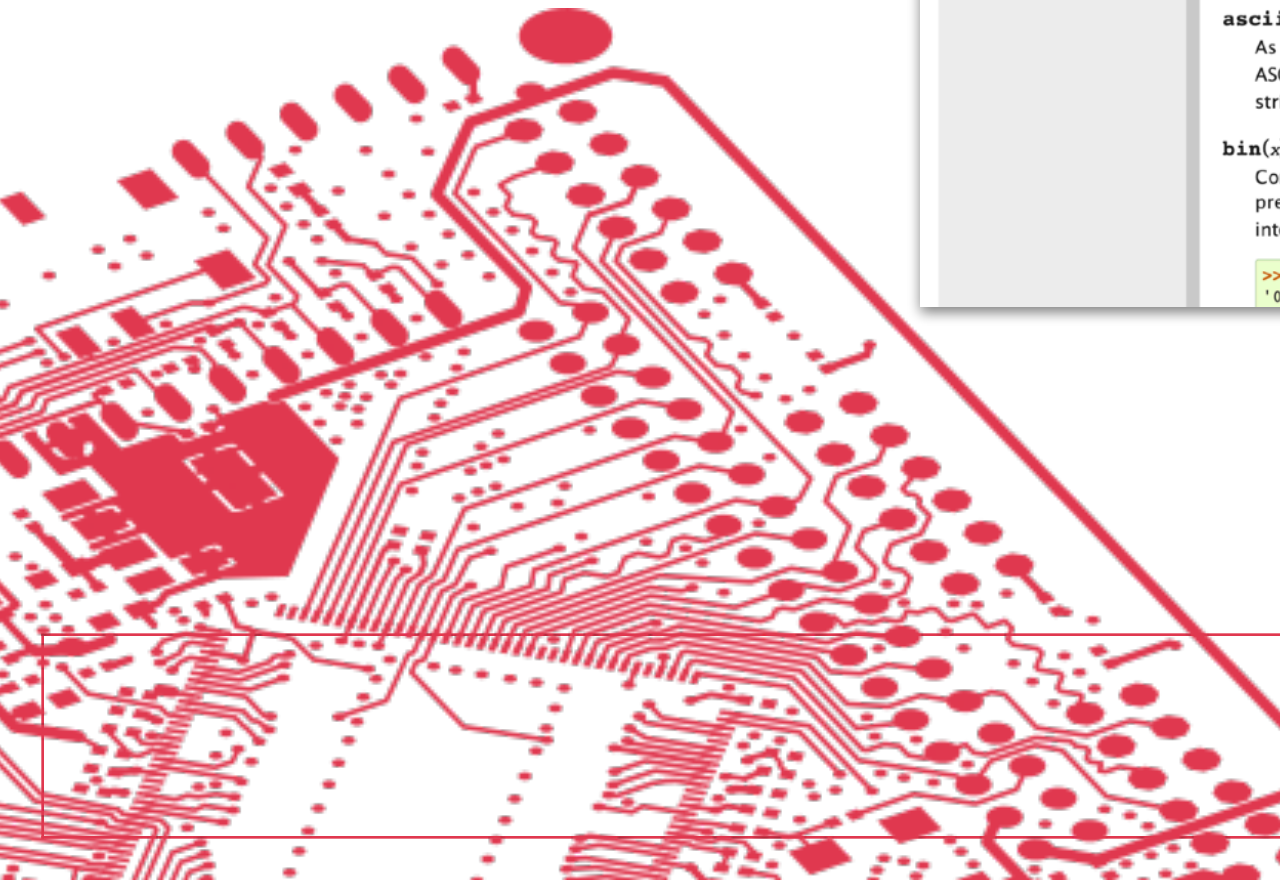
Search

Teams

SCIM

Users

API Status: good





# What is an ABI?

The essence of APIs:

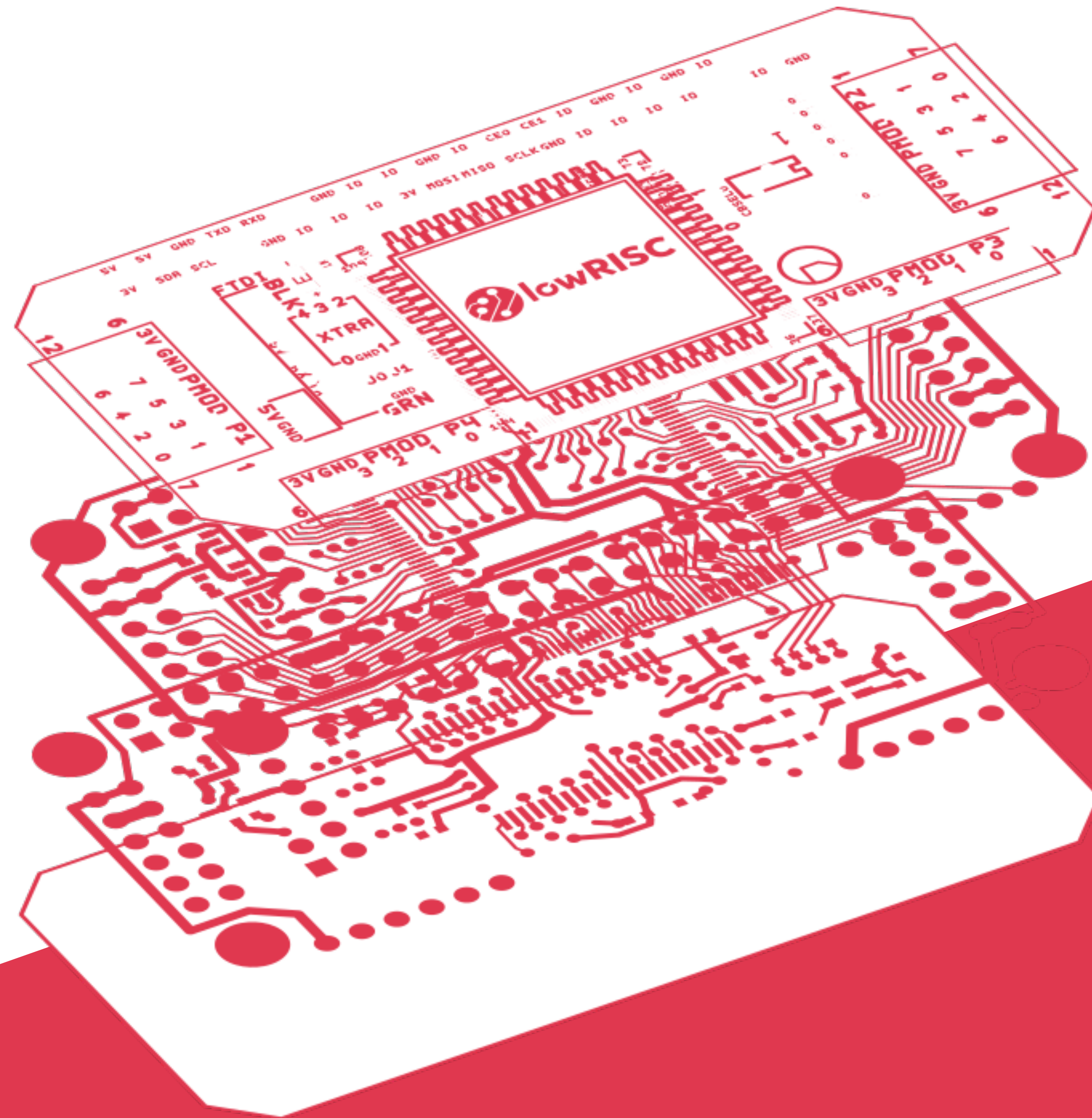
Meaning of an Interface

An ABI is another kind of API

Application *Binary* Interface

A Set of Conventions About:

- Representation of Values
- Where to Locate Items
- How to Achieve Actions





# Why Do We Need ABIs?

N2176

C17 ballot

ISO/IEC 9899:2017

INTERNATIONAL STANDARD

©ISO/IEC

ISO/IEC9899:2017

Programming languages — C

(cover sheet to be replaced by ISO)

This is a working document of SC22/WG14

This version of the document is intended to be the version that is to go into ballot for C17.

— It is based on the transformed E11X version of the document that has been proofread by the members of WG14 and that has been approved by teleconference in June 2017.

— It applies all TCs of closed DRs up to April 2017.

— It applies the changes that have been voted in Markham.

— It updates some normative references.

— It provides the minimal changes required for a new version of the standard.

— It integrates some editorial changes that had been found during the revision process.

A brief explanation of the changes could still be added to the foreword.

Document conventions

This document classifies identifiers into different categories. This categorization is important to produce a correct index.

The classes are

— Normal identifiers, toto.

— keywords, while

— symbols with external linkage of the C library, malloc

— types, size\_t

— predefined macros that alias language features, complex

— other predefined macros, EOF

— pragmas and their particles, STDC

— tag names and members of struct, union or enum, tv\_sec

— name fragments, usually reserved prefixes, atomic\_.

1

er: N4861

2020-04-01

N4849

Richard Smith

Google Inc

ccxeditor@gmail.com

rogramming

Note: this is an early draft, it's known to be incomplete and incorrect, and it has lots of bad formatting.

Compiler A

Compiler

Compiler B

The RISC-V Instruction Set Manual

Volume I: Unprivileged ISA

Document Version 20190608-Base-Ratified


Editors: Andrew Waterman<sup>1</sup>, Krste Asanović<sup>1,2</sup>

<sup>1</sup>SiFive Inc.,

<sup>2</sup>CS Division, EECS Department, University of California, Berkeley

andrew@sifive.com, krste@berkeley.edu

June 8, 2019

	The RISC-V psABI	Sam Elliott	21/05/2020		8
--	------------------	-------------	------------	---	---

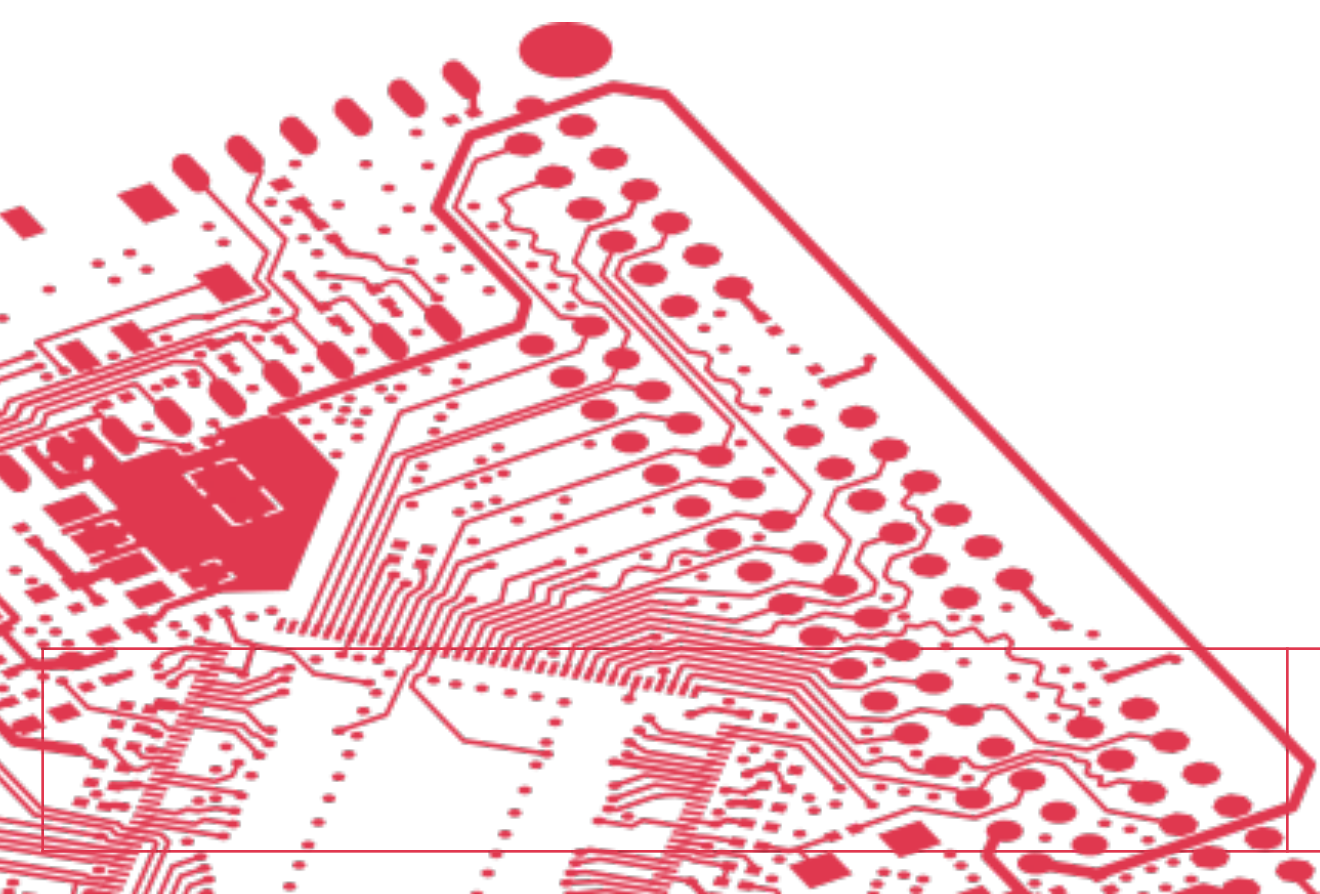


# The RISC-V psABI

## RISC-V ELF psABI specification

### Table of Contents

1. [Register Convention](#)
  - [Integer Register Convention](#)
  - [Floating-point Register Convention](#)
2. [Procedure Calling Convention](#)
  - [Integer Calling Convention](#)
  - [Hardware Floating-point Calling Convention](#)
  - [ILP32E Calling Convention](#)
  - [Named ABIs](#)
  - [Default ABIs](#)
3. [C type details](#)
  - [C type sizes and alignments](#)
  - [C type representations](#)
  - [va\\_list, va\\_start, and va\\_arg](#)





# Where Does The ABI Matter?

```
typedef struct mem_region {  
    char* base_addr;  
    unsigned length;  
} mem_region_t;
```

Types!

Symbols!      Calling Conventions!

```
extern char mem_region_read(mem_region_t region, unsigned offset) {  
    if (offset < region.length) {  
        return region.base_addr[offset];  
    } else {  
        return '\0';  
    }  
}
```



# How Ibex Sees That C Code

```
.text
.globl mem_region_read

mem_region_read:
    bgeu a2, a1, 1f      ; if a2 ≥ a1, jump to 1 (forwards)
    add a0, a0, a2       ; a0 := a0 + a2
    lbu a0, 0(a0)        ; a0 := load (a0 + 0)
    jalr zero, 0(ra)     ; jump to (ra + 0)

1:
    mv a0, zero          ; a0 := 0
    jalr zero, 0(ra)     ; jump to (ra + 0)
```



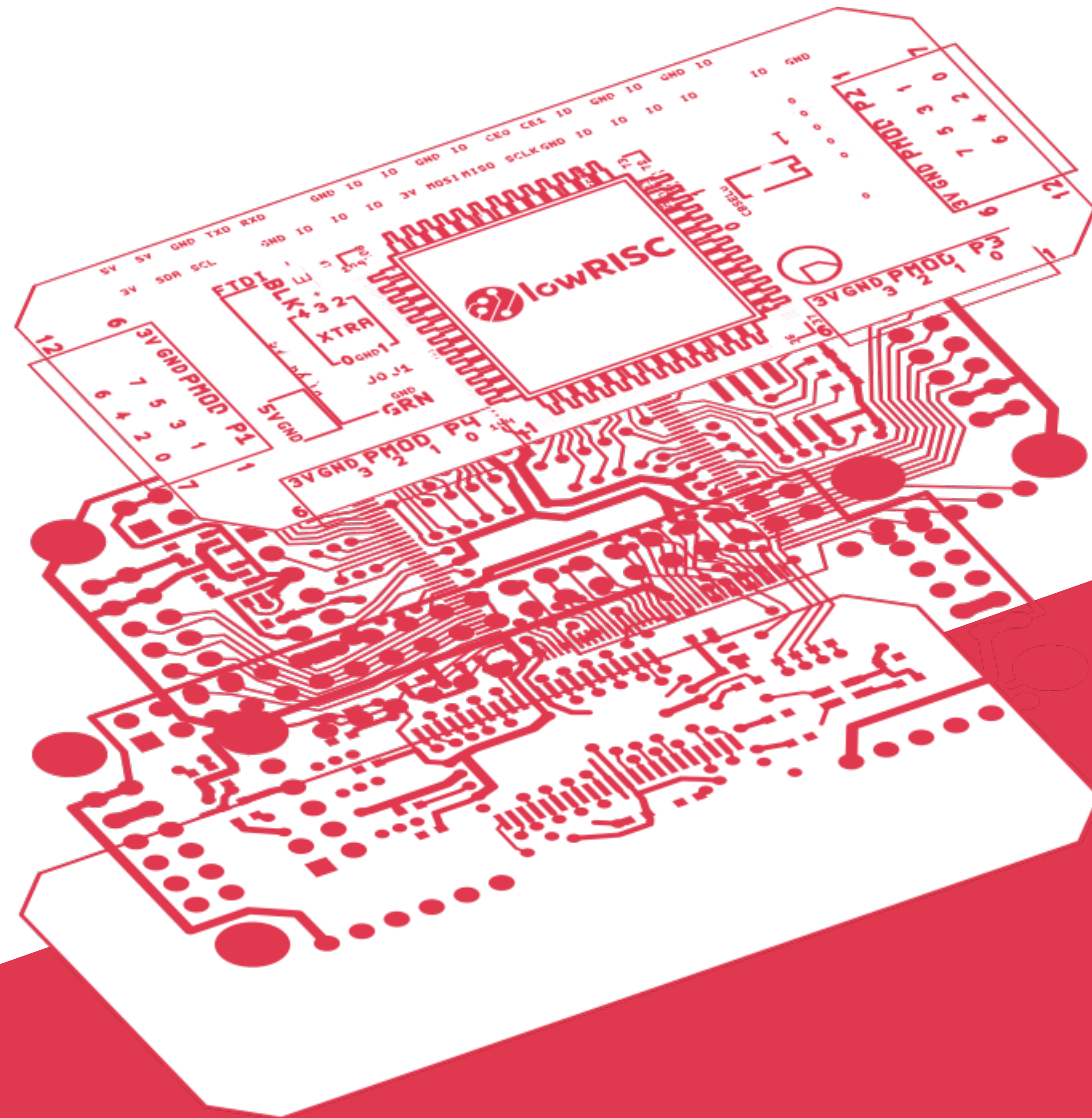


# Types!

How Do We Represent Values?

Major Decisions

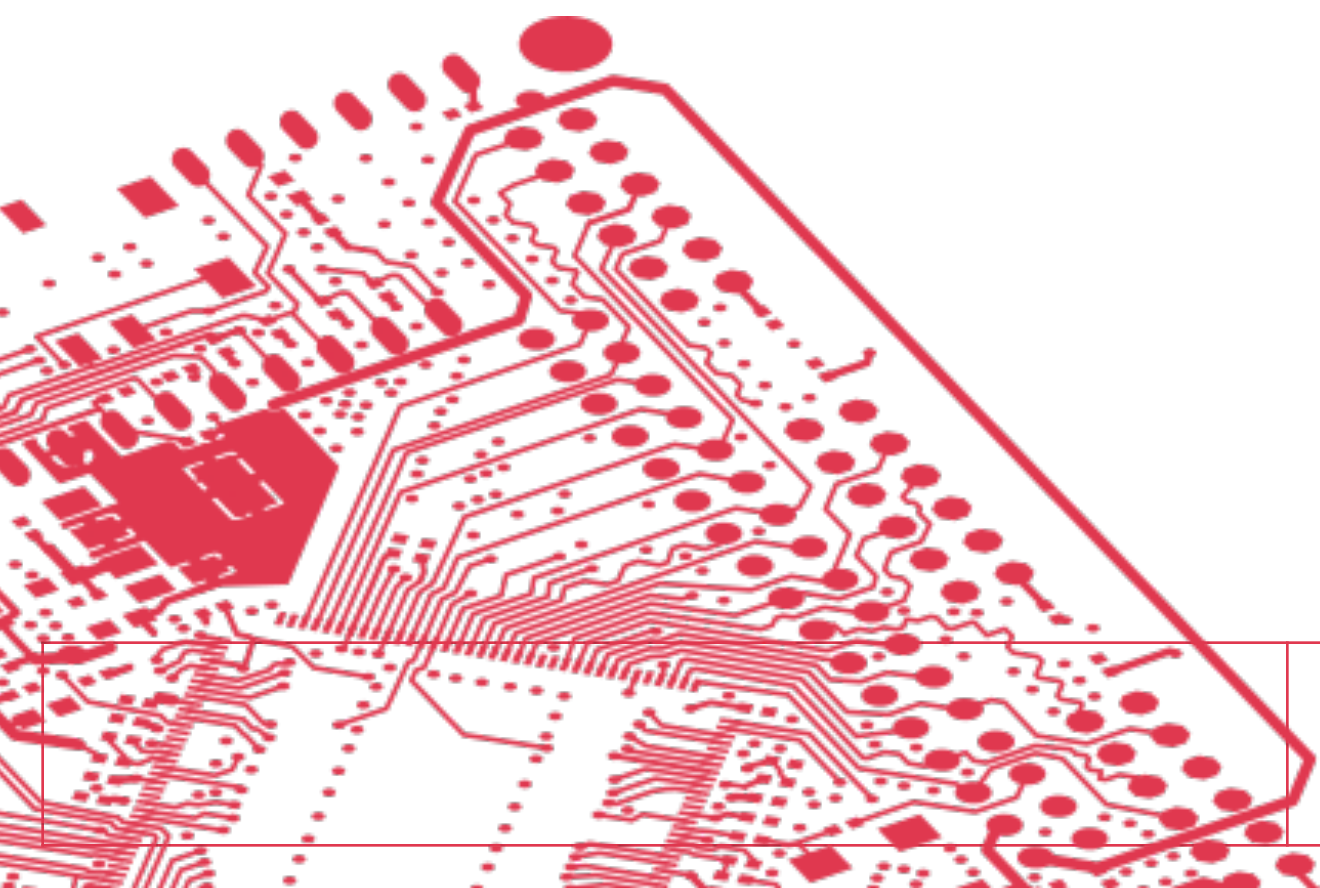
- Sizes
- Signed Values
- Alignment





# Type Conventions

32-Bit	64-Bit
<b>ILP32</b>	<b>LP64</b>
int	long
long	pointer
pointer	





# "Implementation-Defined" and "Undefined"

The RISC-V ELF psABI is a supplement for

- the System-V ABI (including ELF)
- the DWARF Specification
- the Linux Standards Base

... but it also re-uses work:

- the IA32 psABI (x86)
- the MIPS psABI
- the Itanium C++ ABI for IA-64 (Exception Handling)

ISO/IEC 9899:201x Committee Draft — April 12, 2011 N1569

## Annex J (informative)

### Portability issues

1 This annex collects some information about portability that appears in this International Standard.

#### J.1 Unspecified behavior

1 The following are unspecified:

- The manner and timing of static initialization (5.1.2).
- The termination status returned to the hosted environment if the return type of `main` is not compatible with `int` (5.1.2.2.3).
- The values of objects that are neither lock-free atomic objects nor of type `volatile sig_atomic_t` and the state of the floating-point environment, when the processing of the abstract machine is interrupted by receipt of a signal (5.1.2.3).
- The behavior of the display device if a printing character is written when the active position is at the final position of a line (5.2.2).
- The behavior of the display device if a backspace character is written when the active position is at the initial position of a line (5.2.2).
- The behavior of the display device if a horizontal tab character is written when the active position is at or past the last defined horizontal tabulation position (5.2.2).
- The behavior of the display device if a vertical tab character is written when the active position is at or past the last defined vertical tabulation position (5.2.2).
- How an extended source character that does not correspond to a universal character name counts toward the significant initial characters in an external identifier (5.2.4.1).
- Many aspects of the representations of types (6.2.6).
- The value of padding bytes when storing values in structures or unions (6.2.6.1).
- The values of bytes that correspond to union members other than the one last stored into (6.2.6.1).
- The representation used when storing a value in an object that has more than one object representation for that value (6.2.6.1).
- The values of any padding bits in integer representations (6.2.6.2).
- Whether certain operators can generate negative zeros and whether a negative zero becomes a normal zero when stored in an object (6.2.6.2).

554

Portability issues

§J.1



# Calling Conventions

IA-32: All Arguments (and Return Address) are Passed on the Stack.

Why? Only 8 32-bit Registers, some with specific uses. (Stack Pointer uses one)

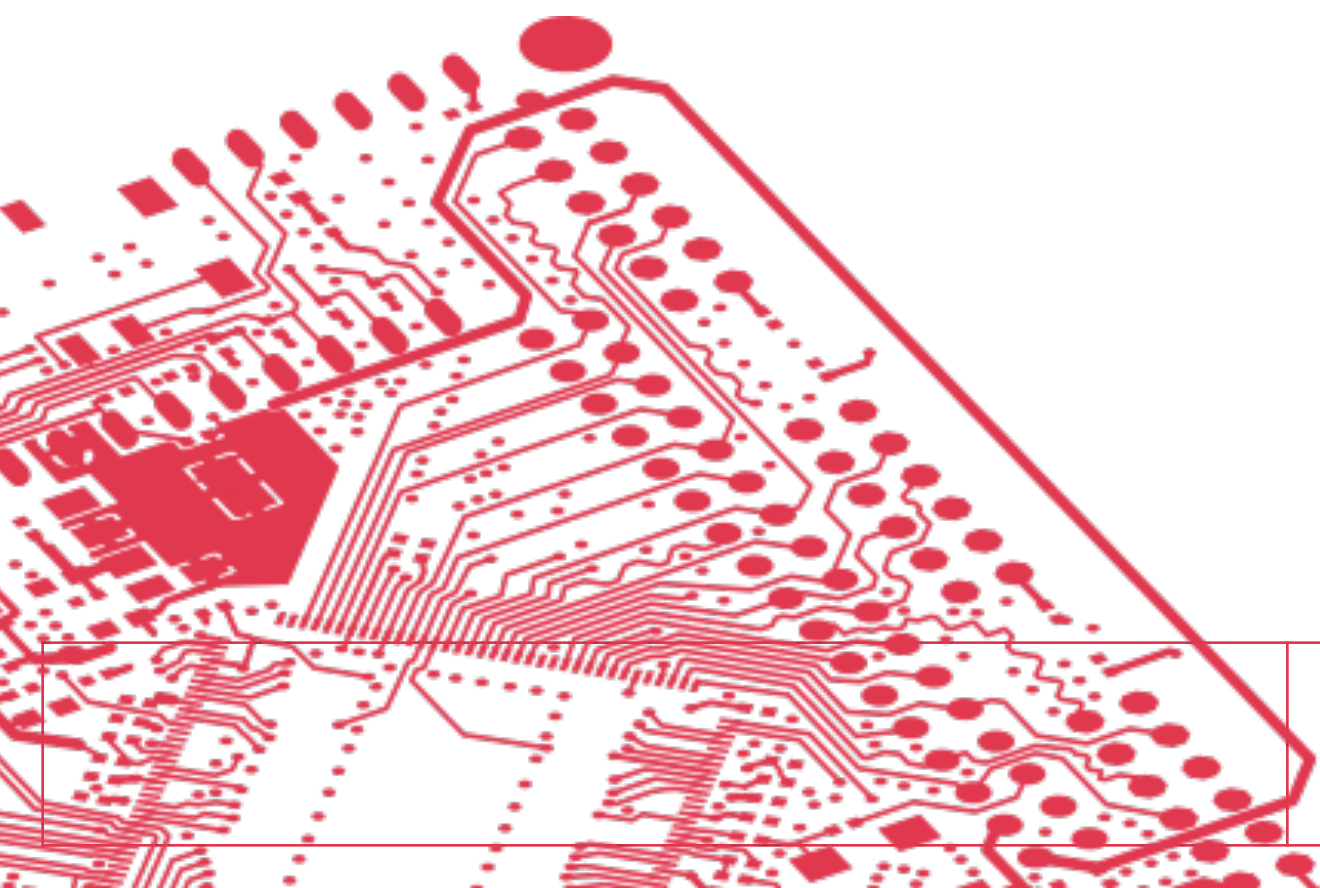
x86-64: First Arguments in 6 Registers, Rest on Stack

Why? 16 64-bit Registers

RISC-V: First Arguments in 8 Registers, Rest on Stack, Return Address in Register

Why? 32 General Purpose Registers

Extra Complexity: Floating Point





# Calling Convention in Action

```
mem_region_read:
    bgeu a2, a1, 1f      ; if a2 ≥ a1, jump to 1 (forwards)
    add a0, a0, a2       ; a0 := a0 + a2
    lbu a0, 0(a0)        ; a0 := *(a0)
    jalr zero, 0(ra)     ; jump to 0(ra)

1:
    mv a0, zero          ; a0 := 0
    jalr zero, 0(ra)     ; jump to 0(ra)
```

```
typedef struct mem_region {
    char* base_addr;
    unsigned length;
} mem_region_t;
```

```
extern char mem_region_read(mem_region_t region, unsigned offset) {
    if (offset < region.length) {
        return region.base_addr[offset];
    } else {
        return '\0';
    }
}
```





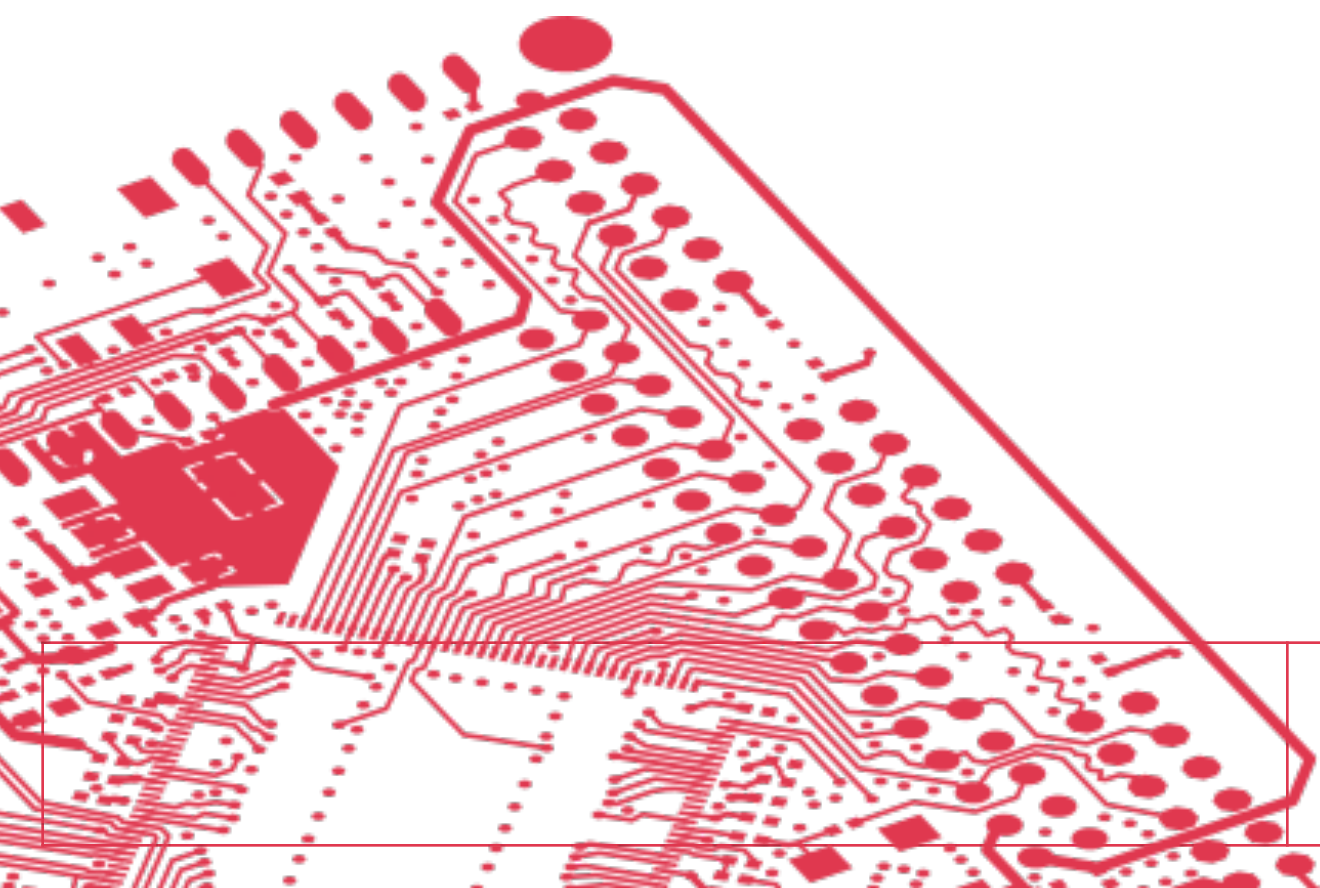
# Stack Management

RISC-V psABI dedicates one register to point to current stack

Interrupts have to respect this convention

Lots of details here are up to Compiler

However: C-extension was Co-designed with the Conventions



# Finding Functions

```
lui ra, %hi(mem_region_read)
    # R_RISCV_HI20(mem_region_read)
jalr ra, %lo(mem_region_read)(ra)
    # R_RISCV_LO12_I(mem_region_read)
```





# Finding Functions

```
label:  
    auipc ra, %pcrel_hi(mem_region_read)  
        # R_RISCV_PCREL_HI20(mem_region_read)  
    jalr ra, %pcrel_lo(label)(ra)  
        # R_RISCV_PCREL_L012_I(label)
```



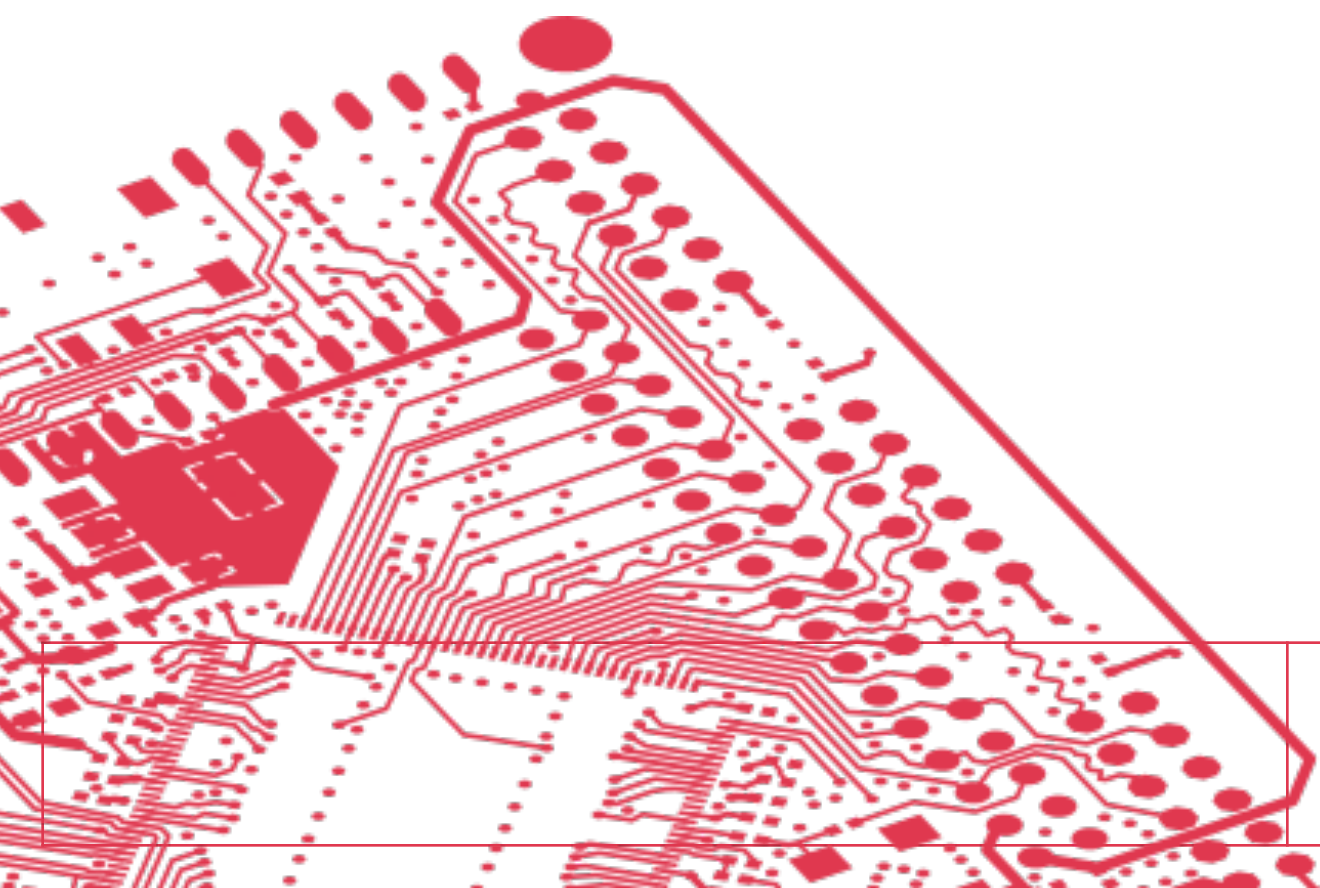
# Code Models

## Monolithic Embedded Images:

- All Addresses Known At Static-Link Time
- Still Allows Relative References

## Linux Objects (With Virtual Memory):

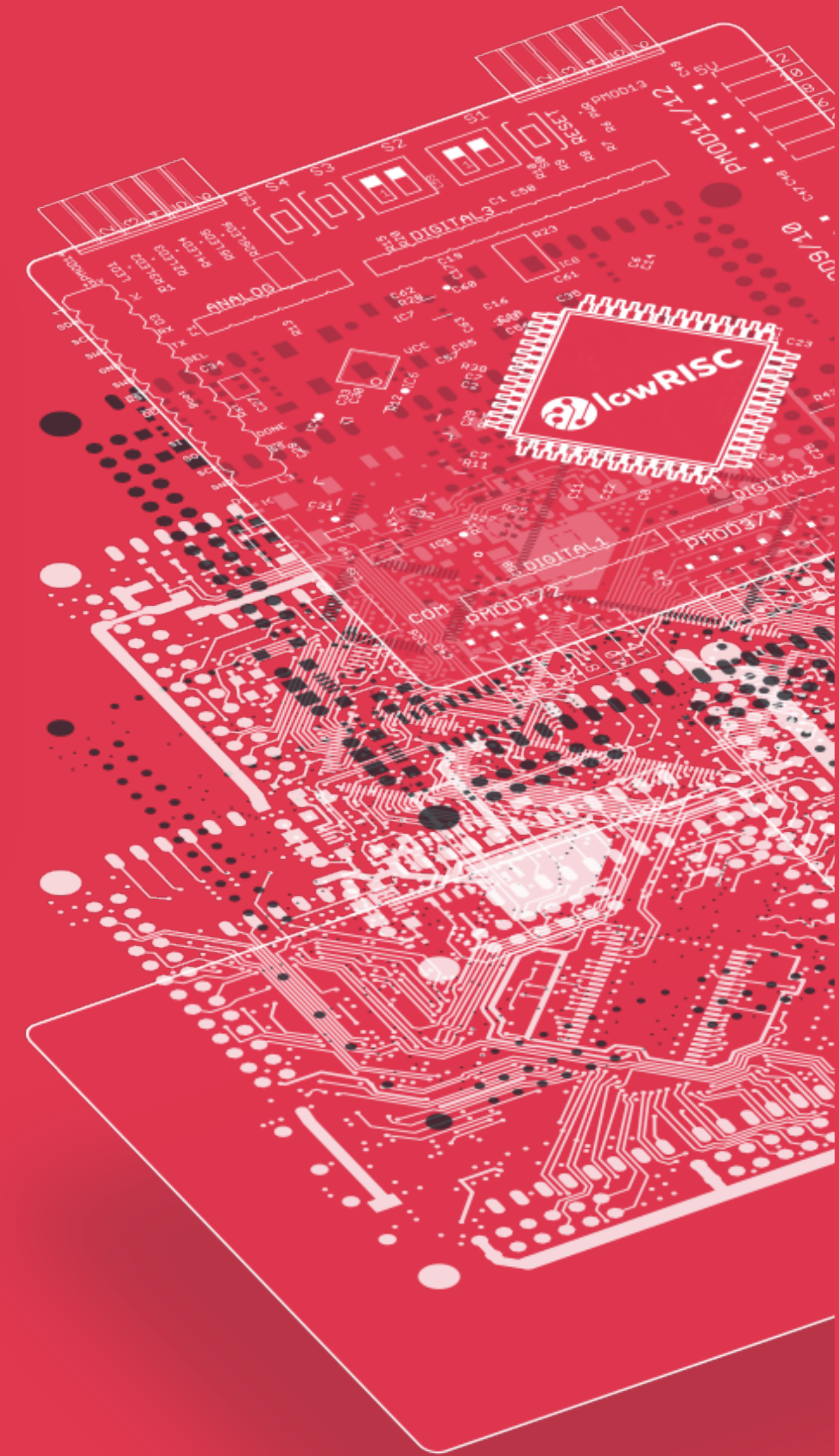
- Relative Offsets Known for Both Code and Data
- Inter-Object References Require GOT or PLT





# Finding Globals

```
label:  
    auipc a0, %got_pcrel_hi(global)  
        # R_RISCV_GOT_PCREL_HI20(global)  
    lw a0, %pcrel_lo(label)(a0)  
        # R_RISCV_PCREL_LO12_I(label)  
    lw a0, 0(a0)
```



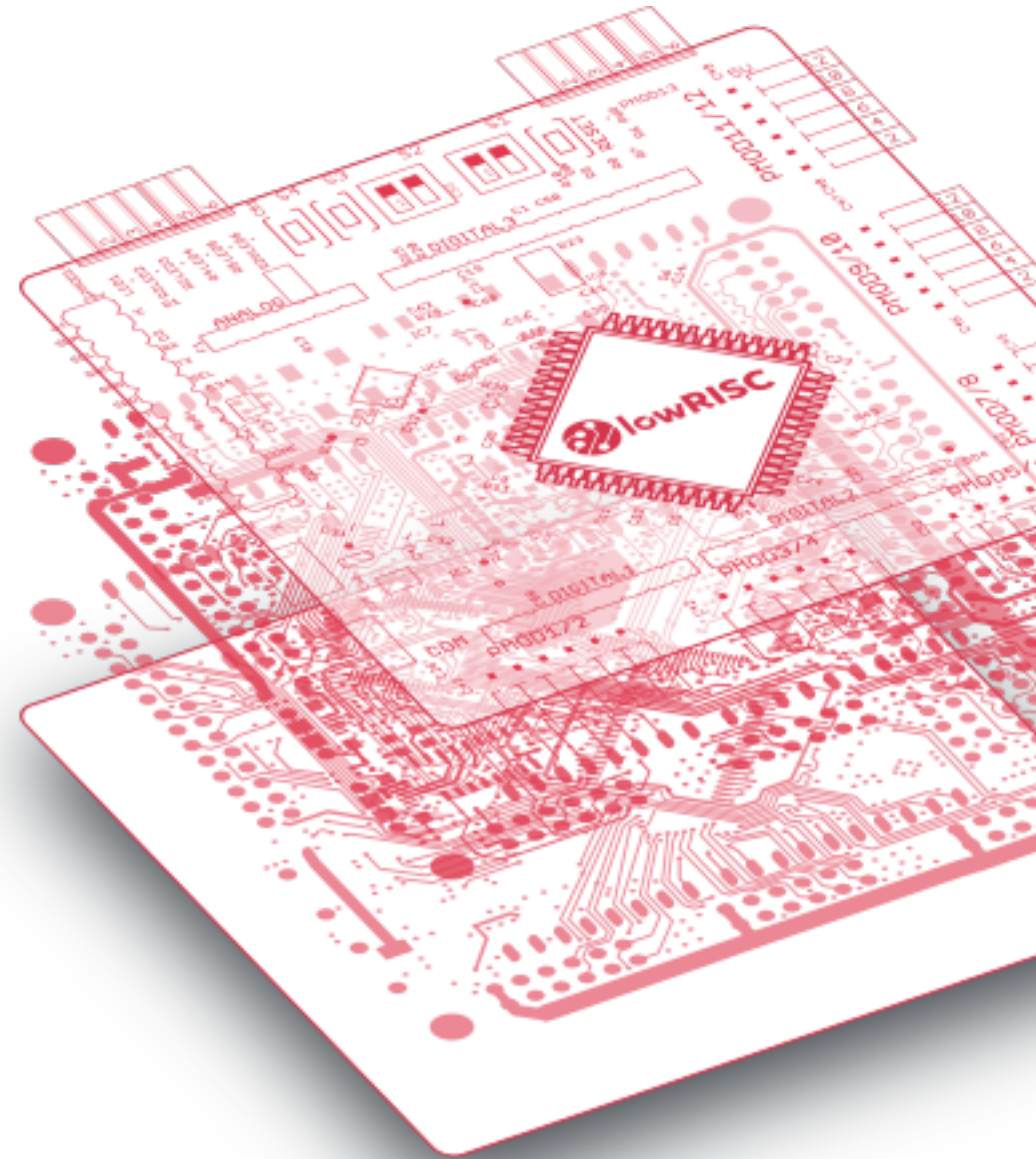


# Linker Relaxation

Do we always need (for example) the upper 20 bits?

- No
- However we only know this at link time.


Linker Relaxation is Peephole Optimisation, using information only the Linker knows.





# Linker Relaxation 1

```
lui ra, %hi(mem_region_read)
# R_RISCV_HI20(mem_region_read)
jalr ra, %lo(mem_region_read)(ra)
# R_RISCV_L012_I(mem_region_read)
```



```
jalr ra, %lo(mem_region_read)(zero)
# R_RISCV_L012_I(mem_region_read)
```



# Linker Relaxation 2

```
label:  
  auipc ra, %pcrel_hi(mem_region_read)  
    # R_RISCV_PCREL_HI20(mem_region_read)  
  jalr ra, %pcrel_lo(label)(ra)  
    # R_RISCV_PCREL_LO12_I(label)
```



```
jal ra, %pcrel_jal(mem_region_read)  
    # R_RISCV_JAL(mem_region_read)
```



# Linker Relaxation 3

```
label:  
    auipc a0, %got_pcrel_hi(global)  
        # R_RISCV_GOT_PCREL_HI20(global)  
    lw a0, %pcrel_lo(label)(a0)  
        # R_RISCV_PCREL_LO12_I(label)  
    lw a0, 0(a0)
```



```
lw a0, %gprel_lo(global)(gp)  
    # R_RISCV_GPREL_I(global)
```



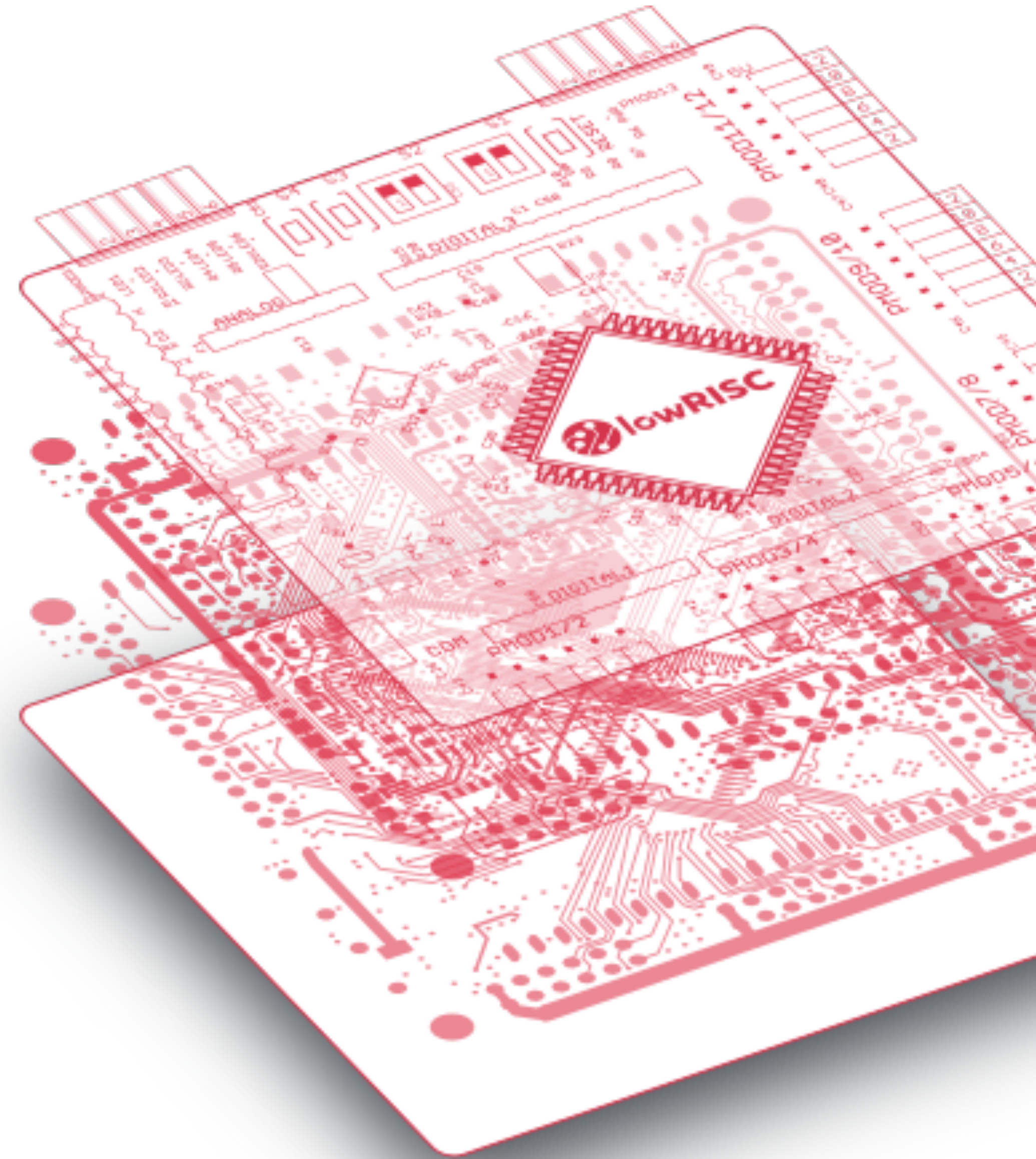
# Linker Relaxation

Helps Code Size: ~5% Savings

Mostly Deletes Instructions

Realigns Instructions

Can Compress Instructions





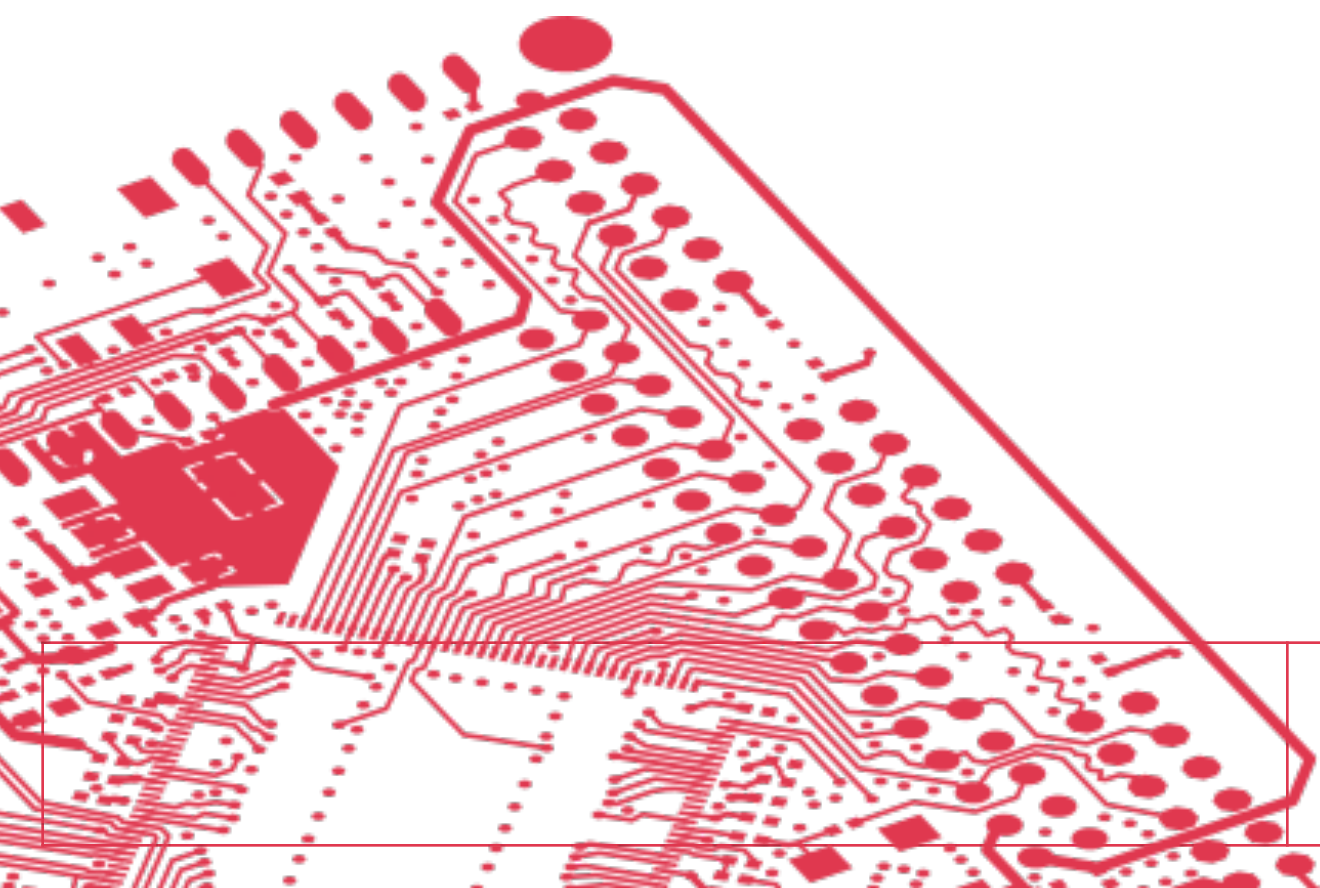
# Program Loading

On Linux:

- OS sets up user-space memory map, GOT
- User-space process does some self-initialization
- Inter-Object calls may need runtime loading

On Embedded Systems:

- The bootloader has to do everything
- No memory map



# Embedded Loading + C Runtime Library

Lots of things need to happen before main can run

- Initialise Data Section
- Zero Uninitialised Data Section
- Initialise Registers (Stack Pointer, Arguments)
- C++ Static Constructors

Relatedly, after main returns

- C's atexit, C++ Static Destructors

```
_start:
.globl _start

// Set up the stack. We have no expectation that the rom that
// jumps here will have the correct stack start linked in.
la sp, _stack_start

// Set up the global pointer. This requires that we disable linker r
// elaxations
// (or it will be relaxed to 'mv gp, gp').
.option push
.option norelax
la gp, __global_pointer$
.option pop

// Set up the new interrupt vector.
la t0, crt_interrupt_vector
csw mtimevec, t0

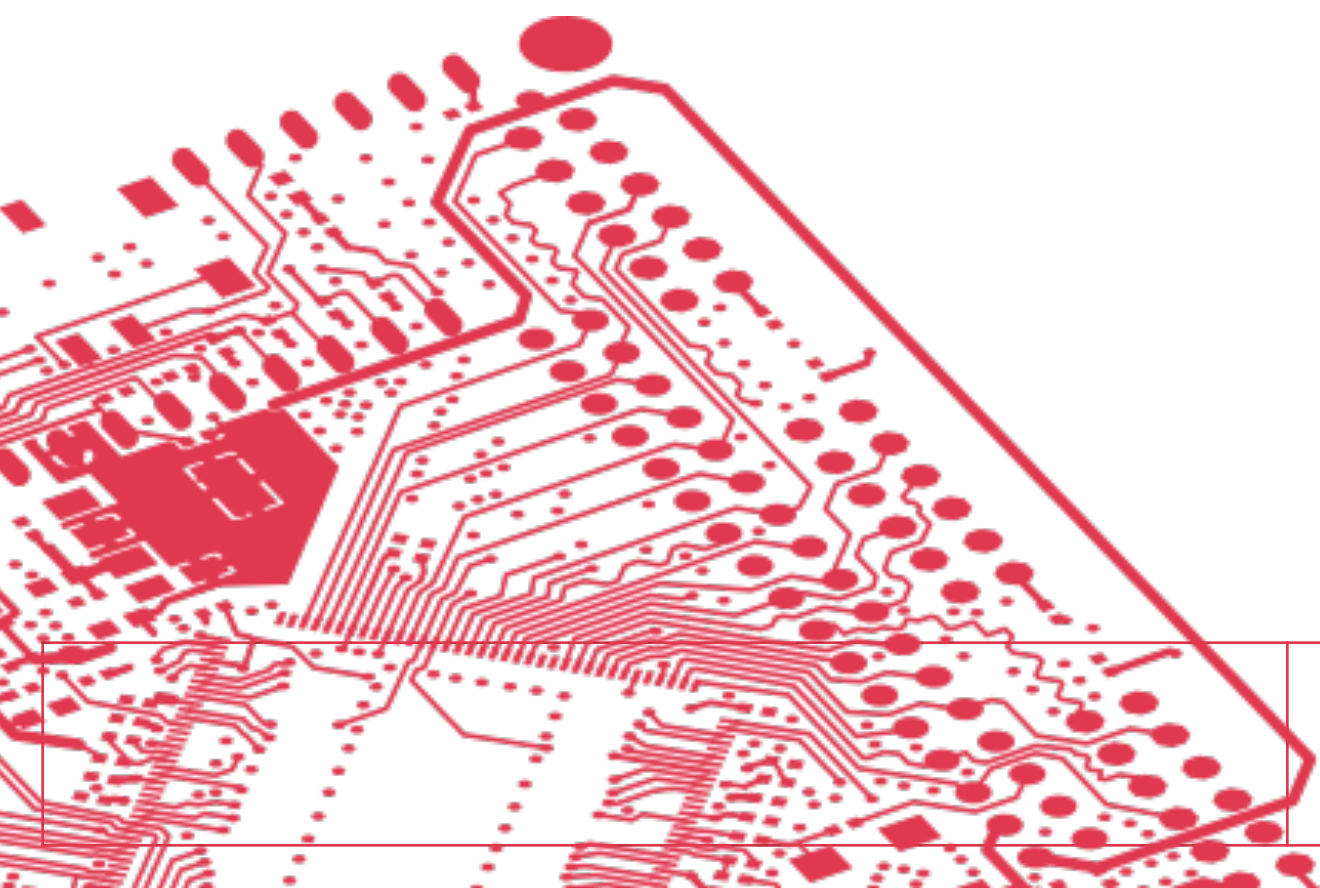
// Zero out the '.bss' segment.
//
// We use 't0' and 't1' to represent the start and end pointers
// of '.bss'.
la t0, _bss_start
la t1, _bss_end
bge t0, t1, bss_zero_loop_end
bss_zero_loop:
sw zero, 0(t0)
addi t0, t0, 0x4
ble t0, t1, bss_zero_loop
bss_zero_loop_end:

// Zero out the stack
//
// We use 't0' and 't1' to represent the start and end pointers of t
// he stack.

// As the stack grows downwards and we zero going forwards the start
// pointer
// starts as _stack_end and the end pointer at _stack_start - 4
la t0, _stack_end
la t1, (_stack_start - 4)
bge t0, t1, stack_zero_loop_end
stack_zero_loop:
sw zero, 0(t0)
addi t0, t0, 0x4
ble t0, t1, stack_zero_loop
stack_zero_loop_end:

// Initialize the '.data' segment from the '.idata' segment.
//
// We use 't0' and 't1' to represent the start and end pointers
// of '.data', 't2' to represent the start pointer of '.idata'
// (which has the same length as '.data') and 't3' is a scratch
// register for the copy.
la t0, _data_start
la t1, _data_end
la t2, _data_init_start
bge t0, t1, data_copy_loop_end
data_copy_loop:
lw t3, 0(t2)
sw t3, 0(t0)
addi t0, t0, 0x4
addi t2, t2, 0x4
ble t0, t1, data_copy_loop
data_copy_loop_end:

// Jump into the C program entry point. This is your standard
// C 'main()', so we need to pass dummy values for 'argc' and 'argv'
li a0, 0x0 // argc = 0
li a1, 0x0 // argv = NULL
call main
```

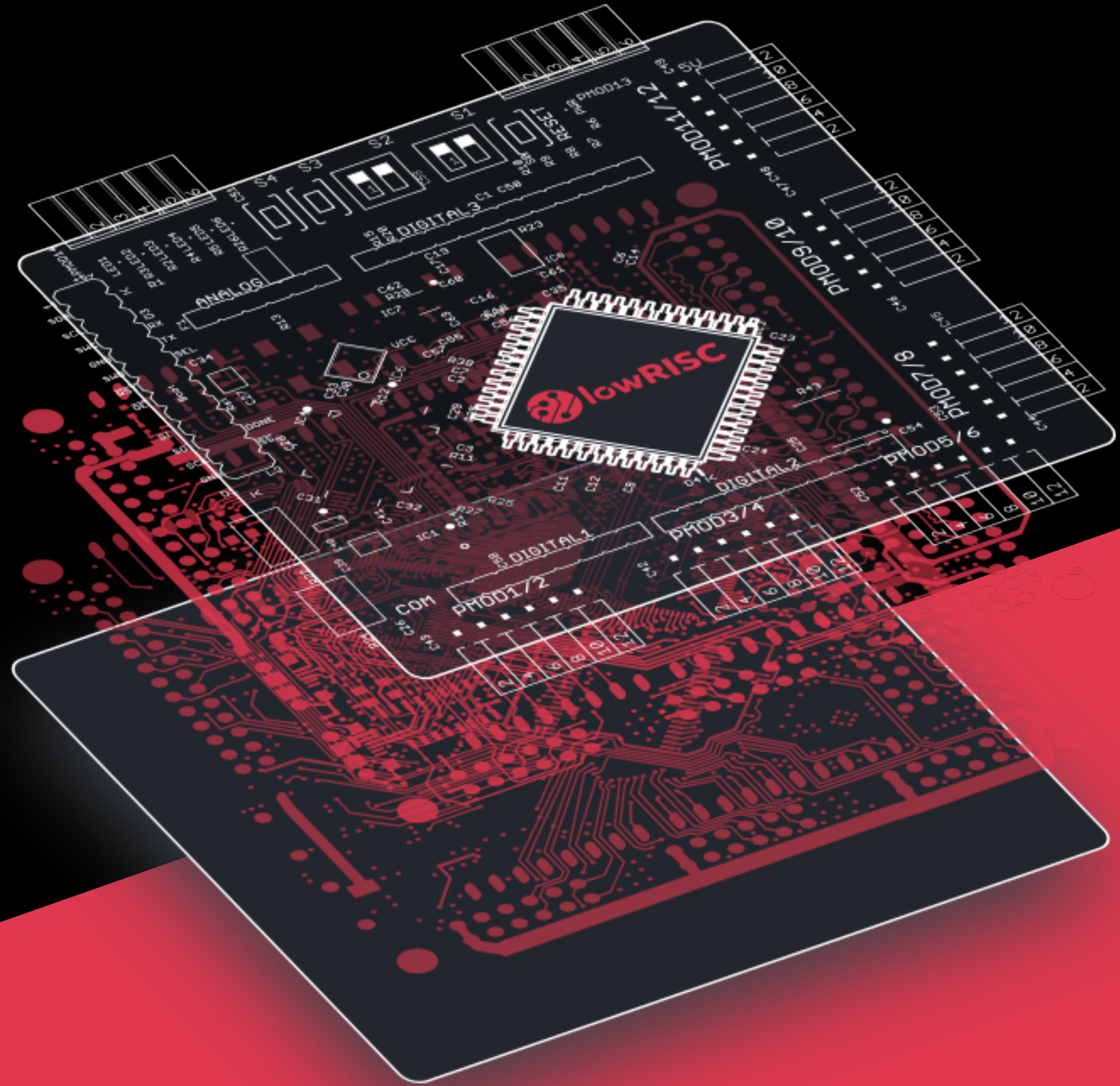




# Further Details

Things I haven't talked about:

- Interrupt Handlers
- Dynamic Loading
- Syscalls and the GNU/Linux ABI
- Floating Point
- Debug and Unwind Information





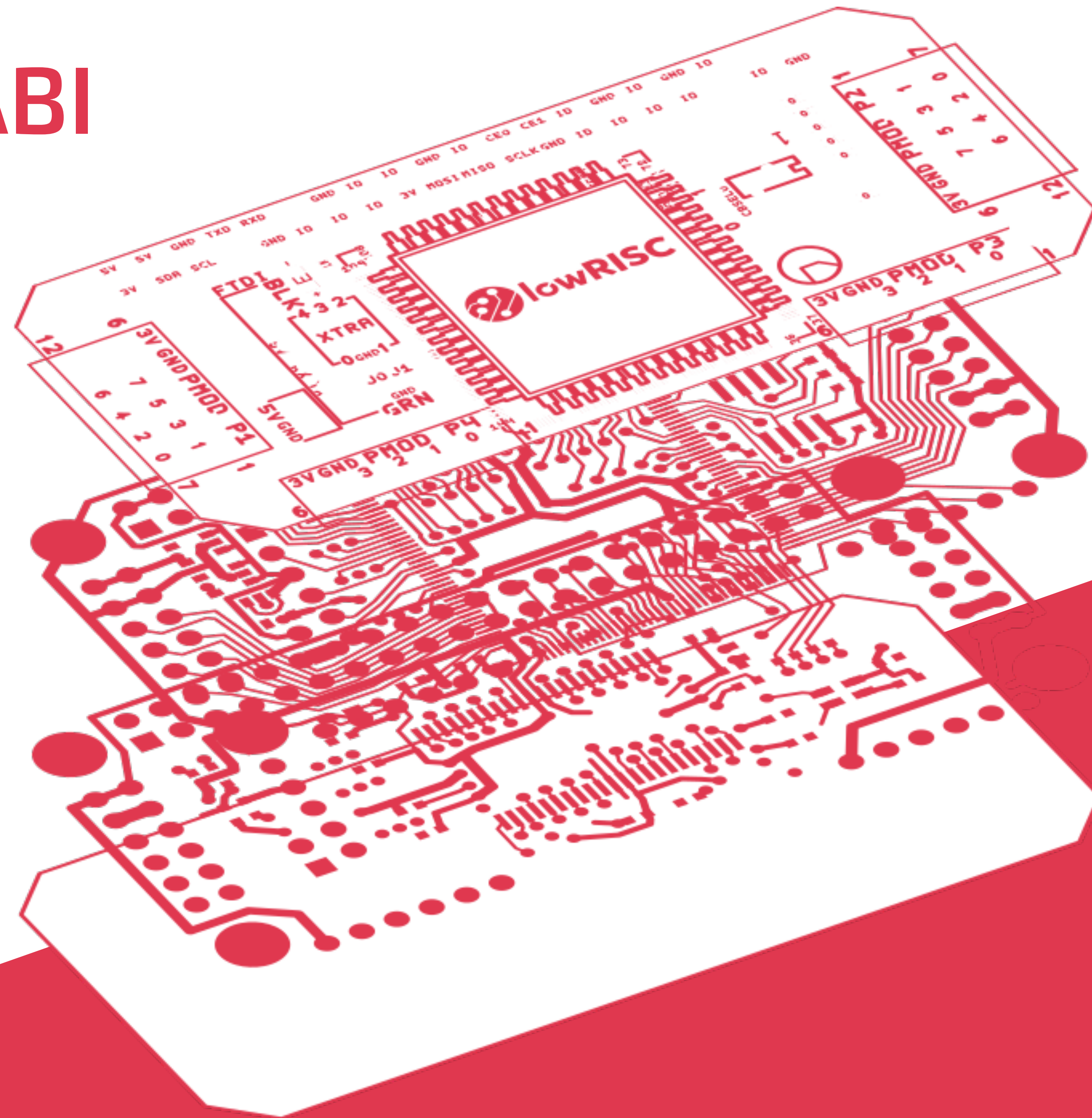
# Future of the psABI

## Embedded PIC

- Function Descriptor PIC
- ROPI/RWPI

## 64-bit Compact Code Model

## Fast Interrupts Working Group





# Embedded PIC

Want to run multiple instances of the same application

It's inefficient to load the code and data each time for each process

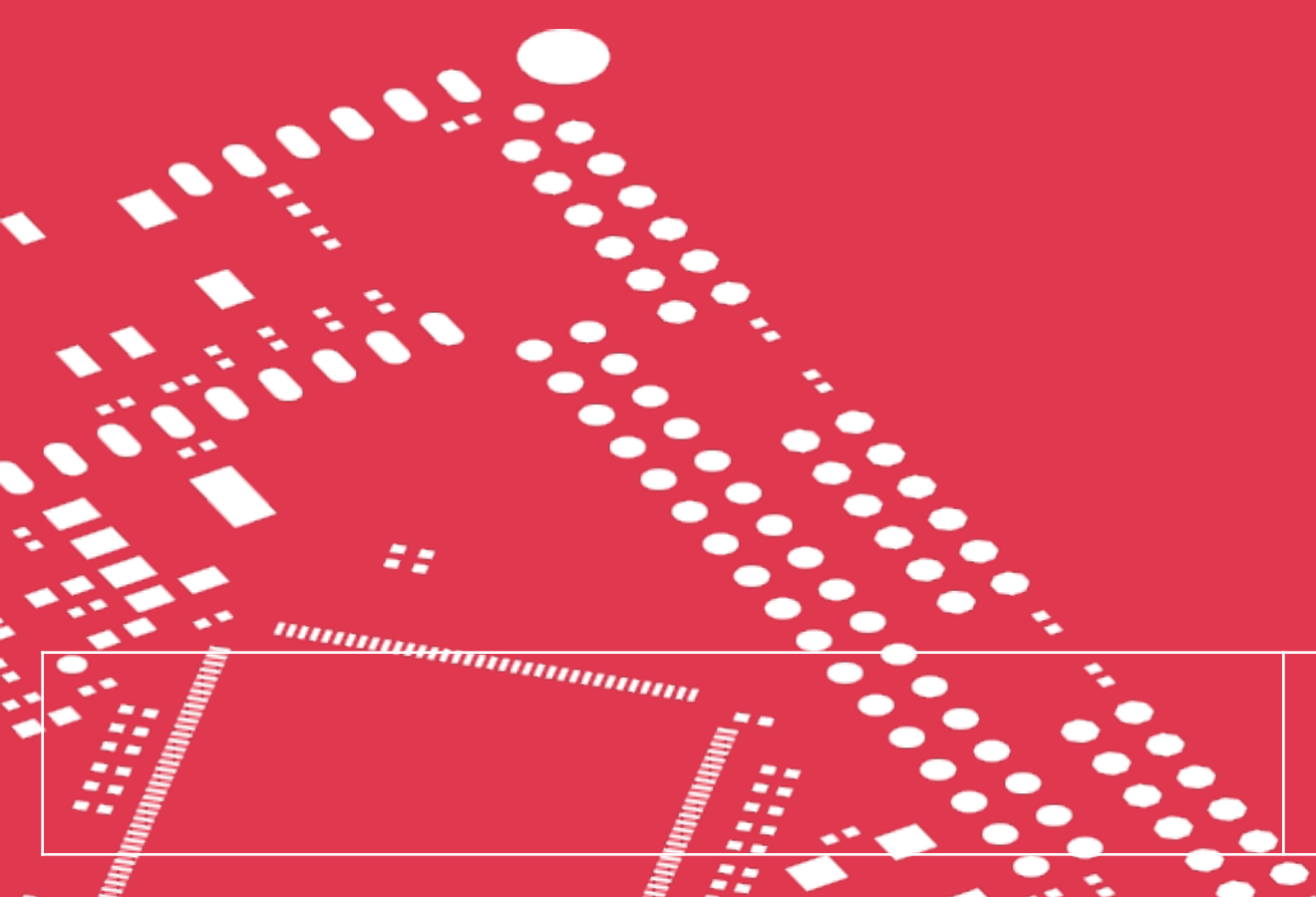
We want to share the code between the processes

No Virtual Memory, so data for separate processes has to be at different addresses

No longer a fixed offset between data and code

Instead, we use GP to help represent this offset

FDPIC supports multiple objects, so each object has its own GP





# ROPI/RWPI for RISC-V

ROPI/RWPI takes a slightly different approach to FDPIC.

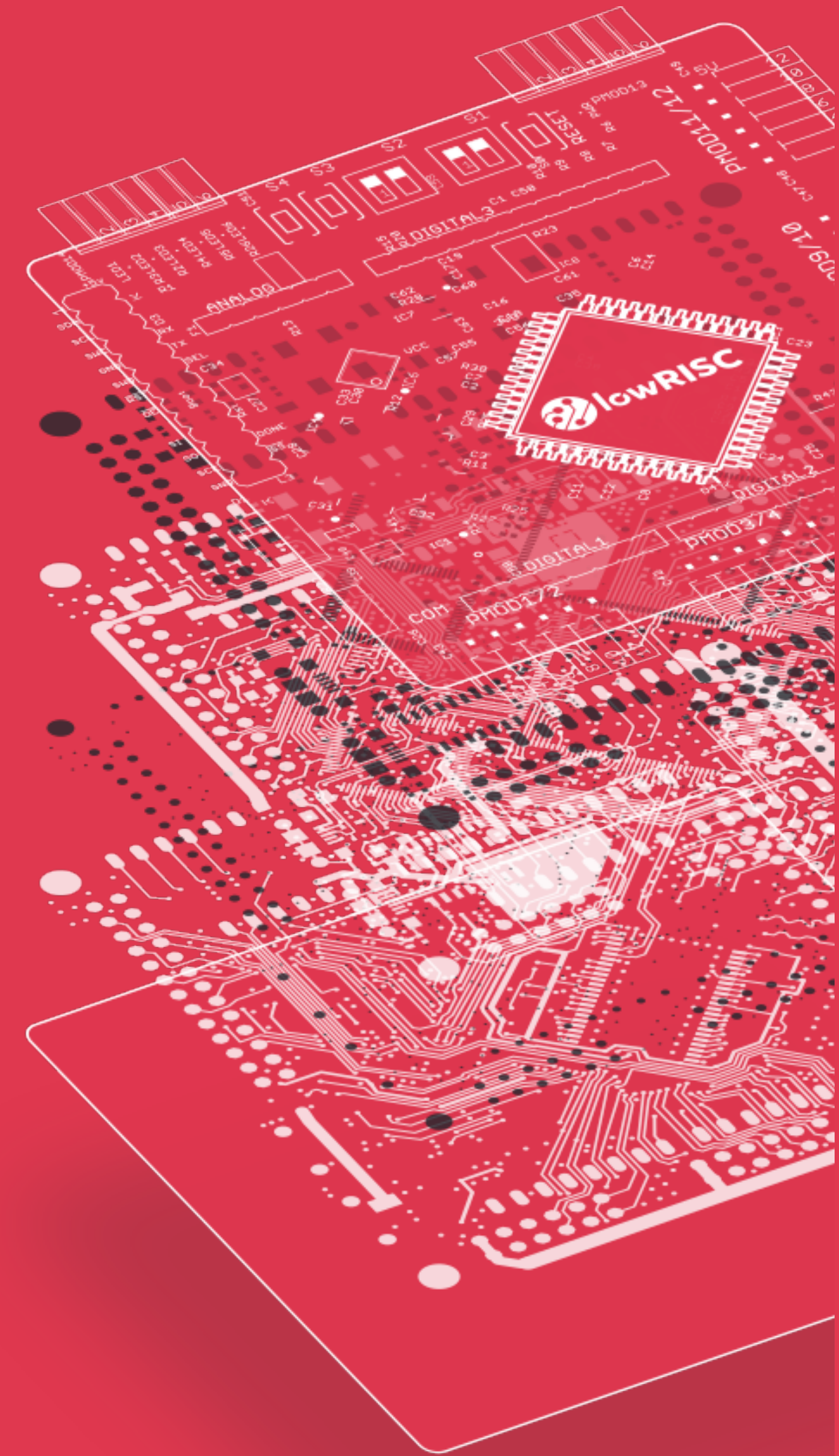
No Dynamic Loading, no GOT, no PLT.

Read-Only vs Read-Write is slightly misleading.

Access to Code and Shared Data is PC-relative.

Access to Private Data is GP-relative.

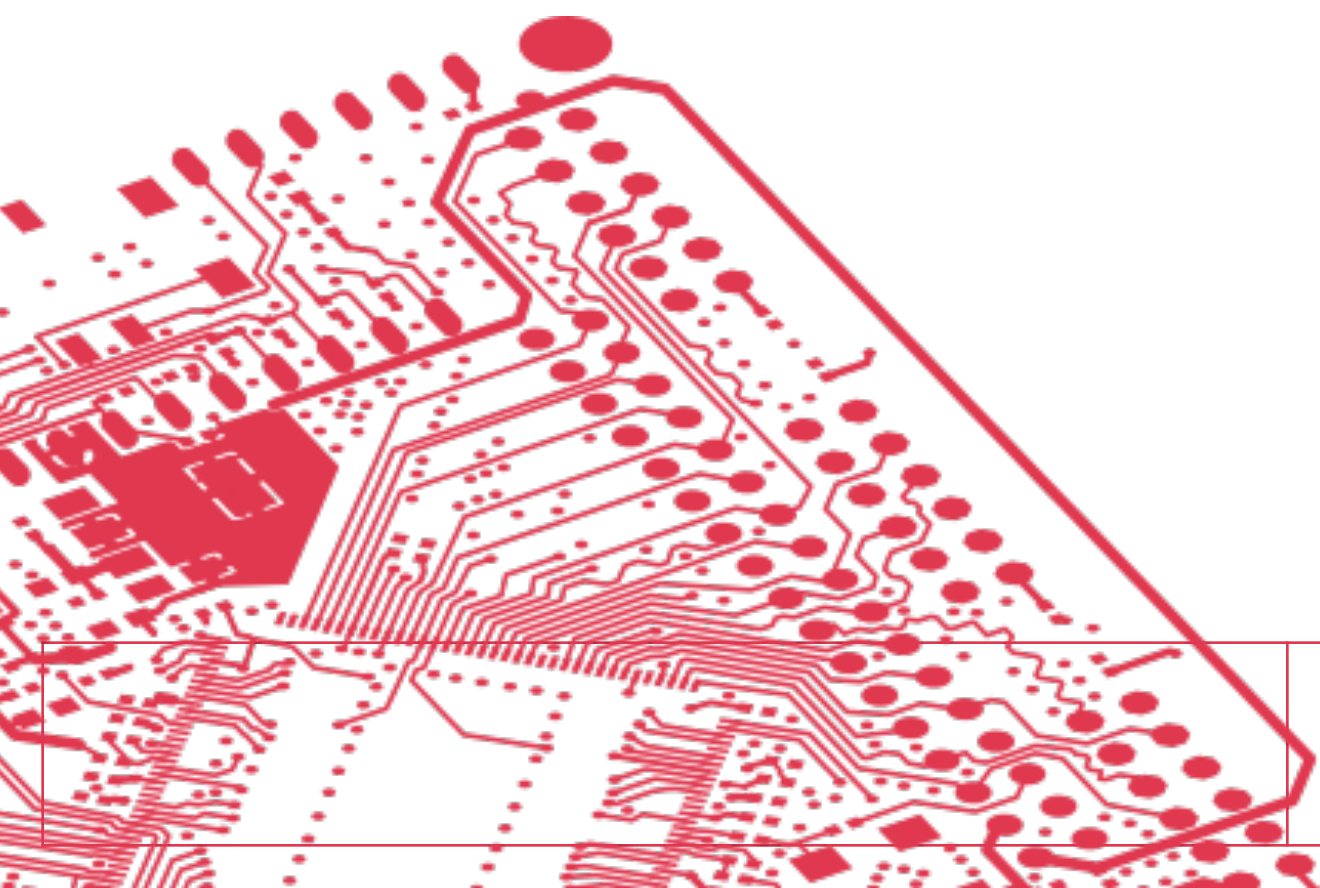
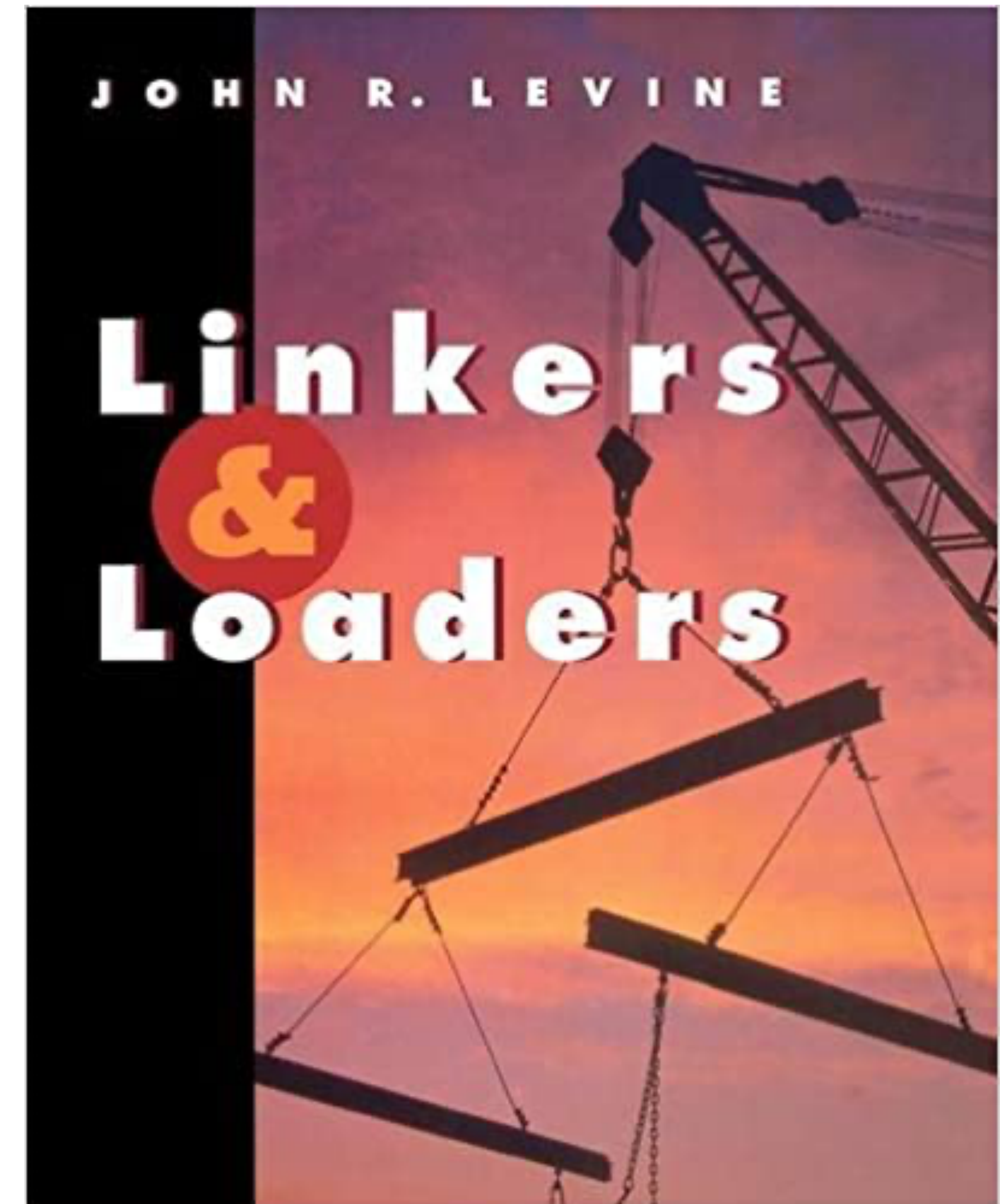
Single GP, so no change to Register Convention.





# Further Reading

- *Linkers and Loaders* by John R. Levine
- SiFive Blog Post on Linker Relaxation





# This Lecture

- An ABI is
- The RISC-V psABI
- Embedded Systems
- Running Programs
- A New Embedded ABI

Any Questions?

