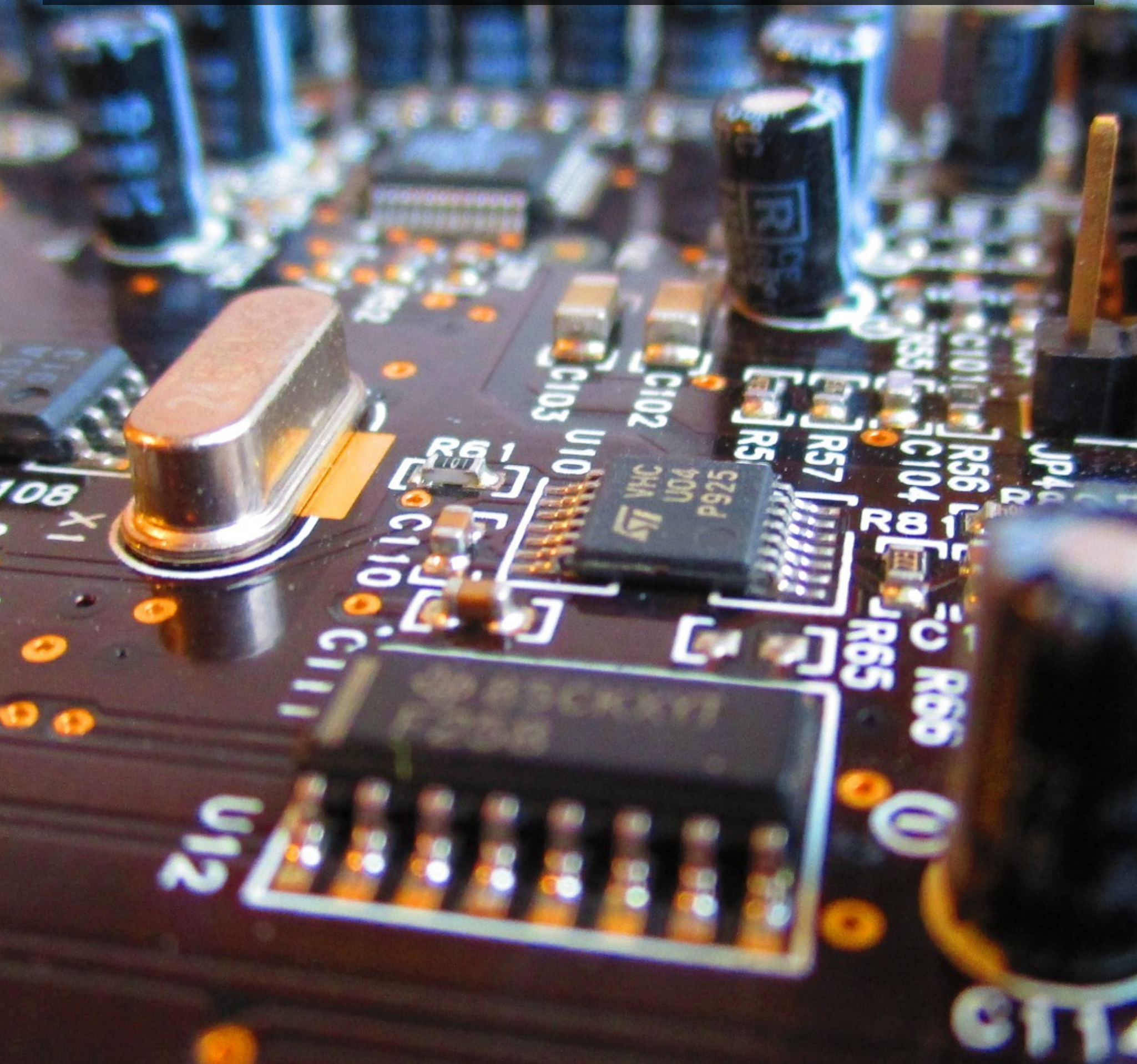


# RISC-V ASSEMBLY LANGUAGE PRIMER FOR ESP32-C3



[MYTECHNOTALENT.COM](https://mytechartalent.com)



**RISC-V Assembly Language Primer for ESP32-C3 .....3**

# RISC-V ASSEMBLY LANGUAGE

## PRIMER FOR ESP32-C3

### Introduction

This primer is designed to give you a foundational understanding of RISC-V assembly language, focusing on the ESP32-C3 microcontroller. By the end of this guide, you'll have a grasp of the core architecture, key registers, and essential assembly instructions needed to start writing your own RISC-V programs.

### ESP32-C3 Architecture Overview

The ESP32-C3 is based on the RISC-V architecture, a streamlined and open-source instruction set. The key components of the ESP32-C3 architecture include 32 general-purpose registers, the Program Counter (PC), and several special-purpose registers.

### Key Registers

- **ra** (Return Address): Holds the return address for subroutines.
- **sp** (Stack Pointer): Points to the current top of the stack.
- **gp** (Global Pointer): Points to the global data section.
- **tp** (Thread Pointer): Points to the current thread context.
- **t0-t6** (Temporary Registers): Used for temporary data that is not preserved across function calls.
- **fp/s0** (Frame Pointer): Points to the base of the current stack frame.
- **s1-s11** (Saved Registers): Used to preserve data across function calls.
- **a0-a7** (Argument Registers): Used to pass arguments to functions and return values.
- **pc** (Program Counter): Holds the address of the current instruction being executed.

## Loading and Storing Words

- **lw** *rd*, **offset**(*rs1*): Load a word from memory into register *rd*.

```
lw t0, 0(sp) # Load word from memory at addr in sp, offset 0 to t0
```

- **sw** *rs2*, **offset**(*rs1*): Store a word from register *rs2* into memory.

```
sw t0, 4(sp) # Store word in t0 to memory at address sp + 4
```

## Arithmetic Operations

- **add** *rd*, *rs1*, *rs2*: Add *rs1* and *rs2*, store the result in *rd*.

```
add t0, t1, t2 # t0 = t1 + t2
```

- **sub** *rd*, *rs1*, *rs2*: Subtract *rs2* from *rs1*, store the result in *rd*.

```
sub t0, t1, t2 # t0 = t1 - t2
```

- **mul** *rd*, *rs1*, *rs2*: Multiply *rs1* and *rs2*, store the result in *rd*.

```
mul t0, t1, t2 # t0 = t1 * t2
```

- **div** *rd*, *rs1*, *rs2*: Divide *rs1* by *rs2*, store the result in *rd*.

```
div t0, t1, t2 # t0 = t1 / t2
```

## Control Flow

- **beq** *rs1*, *rs2*, *label*: Branch to *label* if *rs1* equals *rs2*.

```
beq t0, t1, loop # If t0 == t1, jump to loop
```

- **j** *label*: Unconditional jump to *label*.

```
j end # Jump to end
```

## Subroutines

- **call** *label*: Call subroutine at *label*.

```
call my_function # Call subroutine
```

## Example: Hello World in RISC-V Assembly

Below is a minimal example of using RISC-V assembly within a PlatformIO project to create a simple "Hello World" program.

```
void setup() {  
    Serial.begin(115200);  
    app_main();  
}  
  
void loop() {  
}  
  
void app_main() {  
    int a0_val;  
  
    asm volatile (  
        "li a0, 42 \n"  
        "mv %[val], a0 \n"  
        : [val] "=r" (a0_val)  
    );  
  
    printf("Register a0: %d\n", a0_val);  
}
```

## Conclusion

This primer has introduced you to the core concepts and instructions of RISC-V assembly for the ESP32-C3 microcontroller. With this foundation, you're now equipped to explore more advanced topics and create more complex programs using RISC-V assembly.

You can live test the above example and hack it to try the other instructions that were introduced.

<https://wokwi.com/projects/406420597062574081>

If you are interested in a deeper dive into RISC-V Assembler, please visit the below link.

<https://github.com/mytechnotalent/Hacking-RISC-V>