Programming and Modeling RISC-V on RISC-V architecture with ChatGPT assistance

Przemyslaw Bakowski

LS2N, Nantes University

Abstract

This article presents the didactic and development platform to teach and model RISC-V ISA. Our method is two-fold (software/hardware) and self-contained (modeling RISC-V on RISC-V). The platform itself is largely affordable and running exclusively on open source software, modeling tools included. The initial didactical content is built from four Programming Labs – Plabs, and five Modeling Labs – MLabs. PLabs start with simple examples involving arithmetical instructions and input/output operations. We also delve, with the help of the debugger, into the binary representations to understand the instruction formats and build binary code snippets. MLabs start with a short introduction to Verilog HDL. With the following MLabs we study simple RISC-V architecture, first to model RV32I-subset with R-type instructions then to model full RV32I plus M subsets. Then, running on the RISC-V platform, we inject the generated binaries into the Verilog model. As such the platform is open for further experimentation with RISC-V ISA based programming and modeling. Along with the programming and the modeling processes we specify ChatGPT prompts to generate the test bench codes.

Key-words:
RISC-V ISA, open source hardware-software development platform, assembly programming, Verilog modeling, ChatGPT assistance

Introduction

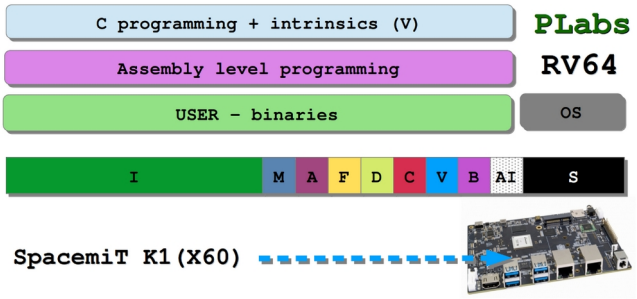
RISC-V is gaining popularity in digital systems due to its simplicity, efficiency, and open-source nature. Understanding RISC-V architecture prepares students for working with these systems. Many companies seek professionals who understand this architecture for its roles in processor design, embedded systems development, IoT systems development, and related fields. Teaching RISC-V architecture via assembly-level programming and RTL modeling allows students to learn core concepts effectively and apply them in practical projects. This initial "programming" stage is important before diving into Verilog coding. It introduces the instruction formats, operation codes, register file structure, memory addressing -access, and control flow mechanisms.

Our two-sided, software-hardware, approach to teach the RISC-V ISA requires the use of an actual hardware platform with RISC-V processor/s.

There are several RISC-V implementations commercially available. The most affordable are the boards/SoCs based on C910 (T-HEAD), JH7110 (SiFive) or K1-V60 (SpacemiT). Our choice is K1-octa-core(V60) SoC integrated on BPI-F3 board [1]. K1 CPU complies with RVA22 profile and 256-bit RVV1.0 standard. The cores are built with eight-stage dual-issue in-order pipeline. BPI-F3 board operates under Debian OS.

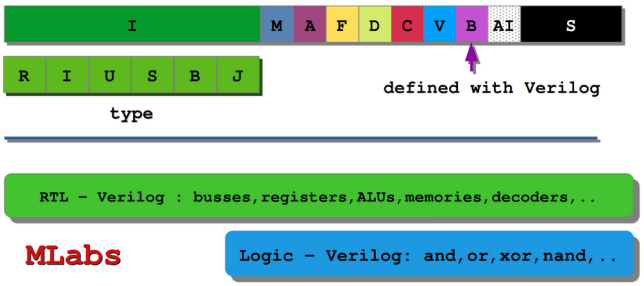
RVLabs

RVLabs are built from two parts, the Programming Labs - **PLabs** related to assembly/binary level programming [2] and the Modeling Labs - **MLabs** related to architectural modeling at RTL level with Verilog HDL [3].



PLabs

Here the aim is to master the essential elements of assembly/binary level programming on RISC-V (RV64) architecture [3]. These include memory layout, essential arithmetic instructions and their binary formats, control instructions, system calls, etc. We also use ChatGPT prompts and the debugger (**dbg**) utilities to generate and analyze the code snippets.



Mlabs

The first two Modeling Labs (**MLab1** and **MLab2**) are related to the use of Verilog HDL. **MLabs** are carried out on the same RISC-V (RV64) platform as the **PLabs**. Instead of C/assembly language and compiler we resort to Verilog language and **iverilog** compiler/ simulator. The test of the simulation results is carried out with **gtkwave** tool. The test bench modules are generated via ChatGPT prompts. **MLab1** introduces the essential features and operators of Verilog and the related tools. In the following **MLab2** we develop a simple "structural" architecture with one processing and one memory blocks (modules). In **MLab3** we develop and test a structural model of **RV32I** architectural subset with only **R-type** of instructions. **MLab4** covers **complete RV32I ISA** architecture with all types of instructions. In **MLab5** we complete this ISA with **M** extension. The following **MLabs** may be used to deal with pipelines and/or further extensions (**FD,V,B,..**).

PLab & MLab example: power_mult.s

First we generate a **RISC-V** program with **assembly macro** named **`power_mult`** which computes **`p=pow(a,b)`** using **repeated multiplication**, and a **test program** that calls the macro with specific registers to demonstrate its functionality.

```
.section .data
result: .word 0 # to store the result
.section .text
.globl _start
# Computes result = base ^ exponent
.macro power_mult result, base, exponent
    li      \result, 1
    beqz    \exponent, end_\@
loop_\@:
    mul     \result, \result, \base
    addi    \exponent, \exponent, -1
    bnez    \exponent, loop_\@
end_\@:
.endm
```

The complete program includes initialization, macro call and termination section. It is assembled and linked with:

$as macro_pow.s -g -o macro_pow.o

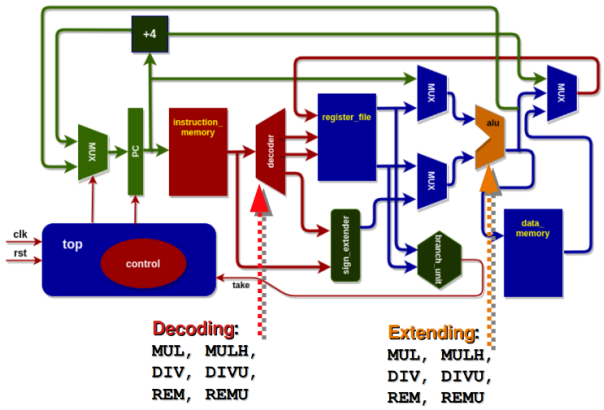
$ld macro_pow.o -o macro_pow

Then we run the debugger:

$gdb macro_pow

```
(gdb) disassemble /r &_start, +56
Dump of assembler code from 0x100e8 to 0x10120:
   0x00000000000100e8 <_start+0>:   00002197
   0x00000000000100ec <_start+4>:   83818193
   0x00000000000100f0 <_start+8>:   00300313    li t1,3
   0x00000000000100f4 <_start+12>:  00400393    li t2,4
   0x00000000000100f8 <_start+16>:  00100e13
   0x00000000000100fc <_start+20>:  00038863
   0x0000000000010100 <loop_0+0>:   026e0e33
   0x0000000000010104 <loop_0+4>:   fff38393
   0x0000000000010108 <loop_0+8>:   fe039ce3
   0x000000000001010c <end_0+0>:    00001297
   0x0000000000010110 <end_0+4>:    01428293
   0x0000000000010114 <end_0+8>:    01c2a023
   0x0000000000010118 <end_0+12>:   05d00893
   0x000000000001011c <end_0+16>:   00000073
End of assembler dump.
```

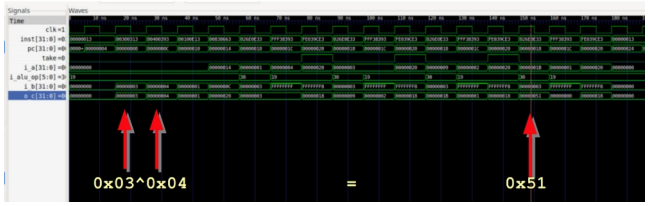
We are now ready to use the prepared binary code for the corresponding modeling example.



The initial model is **RV32I** . Within this model each module is prepared and tested separately. The test bench modules are generated via ChatGPT. We can now extend the model with **M** instructions (**RV32IM**) to run the prepared binary code loaded into **Instruction Memory**. The following is a fragment of an "extended" ALU unit.

```
    // M Extension Operations
    `OP_ALU_MUL:  o_c=i_a * i_b;
    `OP_ALU_MULH: o_c=(($signed(i_a)*$signed(i_b))>>32);
    // Multiplication High (signed)
    `OP_ALU_DIV:  o_c=(i_b!=0)?$signed(i_a)/$signed(i_b):0;
   // Division (signed)
   `OP_ALU_DIVU: o_c=(i_b!=0)?i_a/i_b:0;
   // Division Unsigned
   `OP_ALU_REM:  o_c=(i_b!=0)?$signed(i_a)%$signed(i_b):i_a;
   // Remainder (signed)
  `OP_ALU_REMU: o_c=(i_b!=0)?i_a%i_b:i_a;
  // Remainder Unsigned
  default: o_c = 0;
endcase
```

The compilation and the simulation with **iverilog** generates the reports and waveform to be visualized by **gtkwave** tool. We can see in binary code initializing **macro_pow** at **_start+8** and **_start+12** how to load immediate **3** and **4** (**li** instructions) to the registers **t1/x6** and **t2/x7.**



Conclusion

RVLabs have already been deployed in several engineering courses at Master 2 level. Unfortunately, the limited number of hours to teach Computer Architecture has not enabled to exploit the full potential of the platform.

References

[1] https://docs.banana-pi.org/en/BPI-F3/SpacemiT_K1, 2024

[2] Smith (S.). RISC-V Assembly Language Programming. Apress, 2024.

[3] Cavanagh (J.). – Digital Design and Verilog HDL. – CRC Press, 2008.

[4] Bakowski (P.) - Bakowski (P.) - Teaching RISC-V (ISA) Programming & Modeling on RISC-V platform , "COMPASS" , Nantes, 2024