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

Przemyslaw Bakowski, LS2N

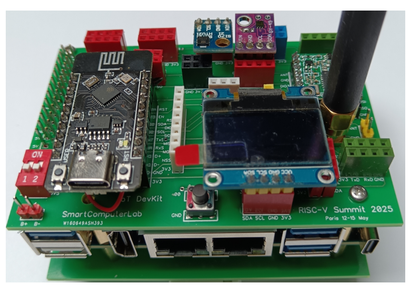
« Verbinden, immer verbinden ! » , Göthe

Introduction

This work 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.

The main board is completed with an extension board to develop RISC-V based IoT Architectures during **RV IoTLabs**. The extension board is based on ESP32C3 SoC with a four-stage RISC-V microprocessor and WiFi/BLE radio modems. The board includes LoRa modems and numerous interfaces for sensors and actuators. This IoT platform is programmed with MicroPython.

Hardware & Software



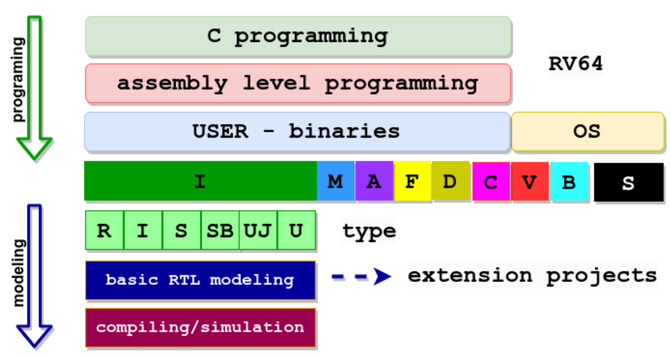
There are several RISC-V (SBC) implementations commercially available. The most affordable ones are the boards/SoCs based on JH7110 (SiFive) or K1/X1 (SpacemiT). Our choice is the X1-octa-core SoC integrated on Orange-Pi RV2 board (OPI-RV2) [1]. X1 CPU complies with the RVA22 profile and 256-bit RVV1.0 standard. These cores are built with an eight-stage dual-issue in-order pipeline. OPI-RV2 board operates under Ubuntu 22.04 OS.

There are three types of software provided for the platform:

- **Programming Labs** use C and assembly languages including debugger functions.
- **Modeling Labs** involve Velilog/SystemVerilog coding and simulation with **iverilog** simulator and **gtkwave** (waveforms viewer).
- Additional **IoT Labs** use Thonny IDE and MicroPython.

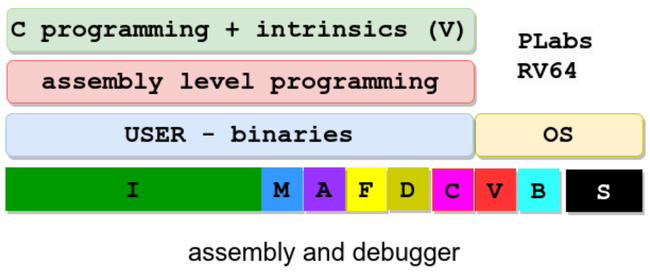
Methodology : RVLabs

Our methodology is based on the programming then modeling RISC-V ISA architectures. Programming Labs - **PLabs** related to **assembly/binary** level programming [2]. Modeling Labs – **MLabs** are related to architectural modeling at **RTL** level with **Verilog** (SystemVerilog)[3]. We generate binary codes representing assembly macros. These codes are in turn used to « program » the Instruction Memory blocks modeled with Verilog at RTL level.



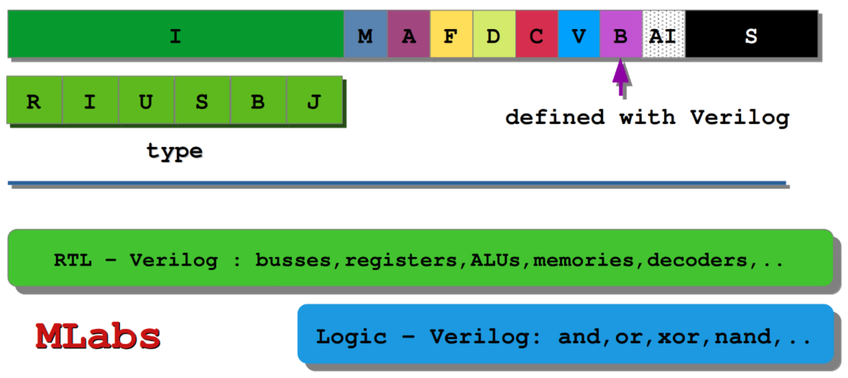
PLabs:

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:

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.



Example : PLab & MLab example with `power_mult.s`

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

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
```

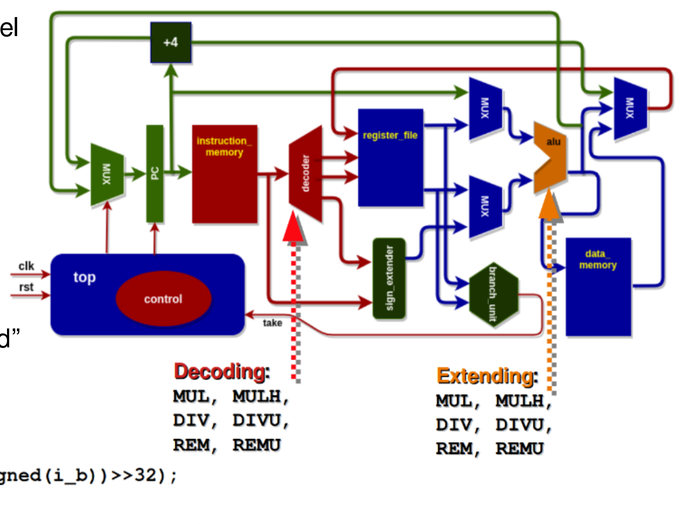
```
.section .data
result: .word 0  # to store the result
.section .text
.globl _start
# Computes result = base ^ exponent
.macro power_mult result, base, exponent
    li      \result, 1
    beqz    \exponent, end_\@
loop_\@:
    mul     \result, \result, \base
    addi    \exponent, \exponent, -1
 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

```
(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.

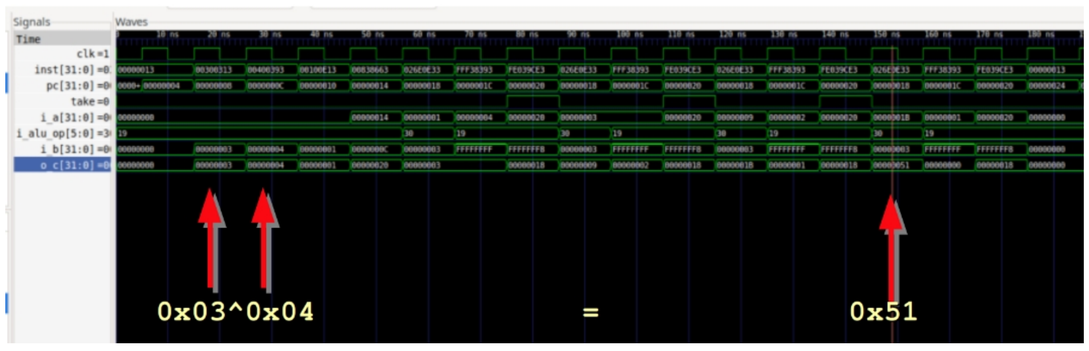
```
// M Extension Operations
`OP_ALU_MUL:   o_c=i_a * i_b;
`OP_ALU_MULH: o_c=(($signed(i_a)*$signed(i_b))>>32);
 // Multiplication High (signed)
 `OP_ALU_DIV:  o_c=(i_b!=0)?$signed(i_a)/$signed(i_b):0;
// Division (signed)
 `OP_ALU_DIVU: o_c=(i_b!=0)?i_a/i_b:0;
 // Division Unsigned
 `OP_ALU_REM:  o_c=(i_b!=0)?$signed(i_a)%$signed(i_b):i_a;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

Decoding:
`MUL, MULH,`
`DIV, DIVU,`
`REM, REMU`

Extending:
`MUL, MULH,`
`DIV, DIVU,`
`REM, REMU`

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.`**



Extension board : IoT Labs

An additional extension board is also based on RISC-V architecture implemented in ESP32C3 SoC. This board is an IoT DevKit connected to the main board. It is programmable « **in situ** » with MicroPython language via Thonny IDE. IoT DevKit provides all essential IoT functionalities and communication means such as WiFi/BLE and LoRa. They are directly available for the designer of IoT Architectures with « pure » RISC-V computing facilities.

Conclusion

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

Acknowledgements

We would like to thank our colleagues and MSc/PhD students for their fruitful feedback concerning numerous courses and tens/hundreds of projects based on our RISC-V platforms.

References

[1] Orange Pi RV2, User Manual, 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.) - Teaching RISC-V (ISA) Programming & Modeling on RISC-V platform , “COMPASS” , Nantes, 2024