Towards a RISC-V Educational HW Lab

V. Mateev, J. Palacios, C. Camarero, B. Pérez, C. Martínez and P. Fuentes*

Departamento de Ingeniería Informática y Electrónica, Universidad de Cantabria

Abstract

Hardware-centric courses at Computer Science and Engineering degrees benefit from using a HW-based approach following an actual computer ISA and an autonomous lab setup for practical sessions. Prior teaching experiences using proprietary ISAs have shown the success of our methodology. However, the RISC-V architecture is an appealing alternative due to its open nature and potential for pervasiveness. This work explores the path towards using RISC-V architecture in the courses and the HW and SW needs that need to be adresssed.

Introduction

Computer Science degrees include a set of HW-centric courses that introduce the student to the behavior of a computer and its interaction with outside devices. These courses allow students to understand the components of a processor architecture and their impact on the performance of a given piece of SW. It is important to place a strong emphasis on the use of practical sessions, where students put the theoretical contents into practice and assimilate the concepts.

HW courses are usually given to beginner students, which are more likely to be encouraged by the use of actual HW. This approach also prevents typical misconceptions favored by the use of simulators. An actual ISA available on the market is more appealing than a fictional architecture, and allows implementing a lab setup for practical sessions based on existing devices. The same principle applies to the learning process of I/O handling, where programming a driver for simple peripheral devices is more engaging than an abstract approach for a fictional device.

At Universidad de Cantabria, the ARM architecture is currently employed for these courses. However, the RISC-V architecture has a promising roadmap due to its Open Source nature, with emerging support in academia and the HPC community. Giving students a first contact with the RISC-V architecture during their most formative years can boost the impact of the ISA. For these reasons, we have set out for a transition to a RISC-V-based environment with features similar to the ones in our current laboratory.

HW and SW Requirements

Introductory HW-based courses call for tools that are easy to use and reduce complexity, while providing sufficient interaction with the HW. For this purpose, we need a HW platform in which students can develop assembly language and I/O programming. We want to avoid the need of an external PC for SW development or *bare metal* configurations. Thus, the HW platform must be able to run an OS, handle regular user peripherals, and provide a general I/O (GPIO) to explore programming drivers.

An OS with a graphical interface is key to ease the learning curve, offering tools such as a text editor with syntax highlighting and a compiling suite. Achieving this requires a GUI-based debugger. Our aim is to adapt our in-house debugger to RISC-V and to an OS that matches the other SW requirements.

A Raspberry Pi-based Lab Setup

When developing the current laboratory setup, the ARM architecture was chosen due to its high market share, which students find more relatable and useful in their foreseeable careers. ARM-based Raspberry Pi board is particularly appealing for its price, availability through retailers, peripheral device ecosystem, and its community of developers. The 1B+ model was selected for its extensive documentation and single-core CPU, which makes the outcome of a program execution more predictable for beginner students.

The *RISC Operating System*[1] was found to very nicely match our needs, as it provides a desktop environment in which we can compile and execute graphical applications. We also appreciate that RISC OS is cooperative multitasking, which makes clear what the machine is executing at each moment. RISC OS further allows user applications to replace interruption handlers and other low level functions, being able to resemble bare metal when desired. We find this very educative, as we generally work on a relatively easy to use desktop environment, and we can show cleanly aspects of the architecture by temporarily removing those SW parts that would interfere.

*Corresponding author: pablo.fuentes@unican.es

This work has been supported by Ministerio de Ciencia e Innovación under contracts PID2019-105660RB-C22, TED2021-131176B-I00, and Ramón y Cajal contract RYC2021-033959-I.

[1] Source code and ISOs can be found in RISC OS Open.

An in-house debugger: UCDebug

The sole limitation observed in RISC OS was the lack of a GUI-based debugger; the default prompt-based debugging tool is very powerful but somehow difficult to use, and GUI-based alternatives were less functional or expensive to acquire. To address this limitation, a debugging tool with a simple, at-a-glance graphical interface was developed in-house. The tool, named UCDebug[2], intercepts CPU exceptions and interrupts to provide the student with pedagogical information about errors and to stop the execution of a program loaded onto the tool. This feature is particularly useful because RISC OS is a cooperative OS that does not preempt a program hoarding computing resources. The use of a graphical interface, made possible by the graphical library provided by RISC OS, allows students to observe at all times memory content and the instructions in their programs. The concrete lab setup developed for the practical sessions and the outcome of the project are described in [1].

A platform for the pandemic

Due to the COVID-19 pandemic restrictions, a remote learning platform was developed with the intent to provide students with a similar experience to the lab setup used in-person. The system, called PiGARDEN[3], allows a student access through a VNC connection to an actual board running RISC OS, and provides the student with controls to handle and to observe those peripherals connected to the board. The limited availability of boards is solved through a time-share scheduler. The system allows students to carry on with their projects through a shared file system, and gives teachers the ability to observe in real time the interaction of a student with any given board. Since its conception, the system has garnered interest from the students due to the lack of affordable devices at retailers spawned by the semiconductor crisis.

Towards a RISC-V HW Lab

Hardware Platforms

We first explored the possible transition to RISC-V in 2019. Some available chips at that time were SiFive's HiFive1 and HiFive Unleashed. The HiFive Unleashed was closer to the Raspberry Pi in our current lab setup, but was dismissed due to budget constraints in favor of the HiFive1. HiFive1 limitations forced offsetting development tasks to a regular PC, and sending back to the device for execution, opposing the needs of the desired teaching environment.

Nowadays there are far more affordable RISC-V chips which could adequately replace the use of the Raspberry Pi. There is the SiFive VisionFive 2[4] which, although above the price bracket of a Pi board, is significantly less expensive than a regular PC and is equipped with up to 8GB of RAM and a 64-bit 1.5GHz CPU. A less expensive alternative is the Sipeed Lichee Pi 4A[5], which uses a 64-bit CPU and can be configured with up to 16GB of RAM. Both boards have an HDMI interface, two Gigabit Ethernet ports, USB connectivity and a GPIO connector with 20 and 40 pins for the Lichee and VisioFive boards, respectively. Another option is the Asus Tinker V board[6], intended for IoT purposes and less powerful, with a single-core processor and 1GB of RAM. This board comes from a more established manufacturer, but is not yet available. We are currently evaluating the VisionFive 2 and Lichee Pi boards, to assess their suitability for our teaching needs.

Software Suite

In our first exploration of RISC-V we worked bare metal on the HiFive1 board while developing in a PC. In the PC we employed SiFive Freedom E SDK Toolchain to build and debug, and the PlatformIO extension of Visual Studio Code to upload code to the HiFive1 board. This showed that executing and debugging programs in a RISC-V board was feasible, but still required an external PC. We also managed to install Zephyr OS on the HiFive1, but it was not further explored.

OS availability for RISC-V has risen as the platform garners interest and adoption, although OS support varies between devices. The boards that are being analysed support general Linux distros, such as Debian or Fedora, and purpose-built platforms such as Tina[7] or Waft[8].

Our next step is to find the desired SW features in an OS that can be booted in a RISC-V board.

References

[1] Pablo Fuentes et al. "Addressing Student Fatigue in Computer Architecture Courses". In: *IEEE Transactions on Learning Technologies* 15.2 (2022), pp. 238–251. DOI: 10.1109/TLT.2022.3163631.

[2] Available in a public GitHub repository.

[3] The SW architecture and a description of the HW configuration can be found at `https://gitlab.com/pigarden`

[4] https://www.kickstarter.com/projects/starfive/visionfive-2

[5] https://sipeed.com/licheepi4a

[6] https://tinker-board.asus.com/product/tinker-v.html

[7] https://github.com/Tina-Linux

[8] Web Assembly Framework for Things. https://chuangke.aliyun.com/waft