Automatic testing of RISC-V assembly programs

Helder Daniel

Departamento de Engenharia Electrónica e Informática
Faculdade de Ciências e Tecnologia
Universidade do Algarve

December 27, 2021

Abstract

RISC-V is an open standard reduced instruction set computer (RISC) instruction set architecture (ISA). RISC-V International, which has major industry players as partners, manages the development of RISC-V specification ([2021]). Recently RISC-V specifications has taken a lead role in the industry, being a must in computer architecture training programmes. This work develops a Mooshak programming contest manager (Leal and Silva [2003]) extension to perform black box testing on RISC-V assembly programs, using the freely available RARS, a RISC-V Assembler and Runtime Simulator (Landers [2021]).

1 Introduction

RISC-V open standard RISC ISA is quite recent but is supported by major industry players, making it a must in Computer architecture training programmes. Although there are available chips in the market, it is still uncommon to have it at the core of personal computers. One way to develop RISC-V applications without having the actual RISC-V hardware is to use a cross compiler, that generates code for an arbitrary target processor. However it is still needed the actual RISC-V hardware to run it. Another way is to use a simulator, that includes the cross compiler or assembler and also a simulated RISC-V environment where the generated applications can be ran and tested.One popular simulator for RISC-V is RARS, RISC-V Assembler and Runtime Simulator (Landers [2021]). It can simulate the basic RISC-V specifications ([2021]).

Using RARS simulator, and taking advantage of Mooshak (Leal and Silva [2003]) programming contests manager, software submission management and basic black box testing features it is possible to automatize the process of testing RISC-V assembly programs.
Mooshak was designed to manage contests, according to ICPC rules (ICPC [2019]), on the web, thus having a web portal where users can submit software to be tested, performs basic black box testing and sends to the user feedback on that testing.
However it has a fundamental limitation undesirable for testing more than very simple applications. The source code must be submitted in just one file. While this can be suitable to manage programming contests it is unsuitable to test real world applications where source code is clearly organized and distributed by many files.

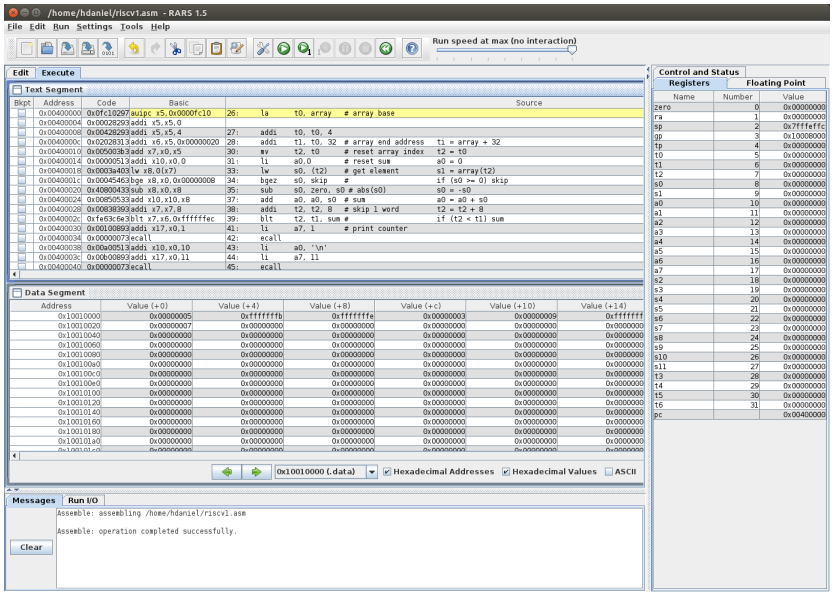

Figure 1: RARS simulator

This work proposes a Mooshak extension to support the testing of real world RISC-V assembly programs. The assembly and running of the executable program is done inside the RARS simulator. Code injection is also supported, making the testing process more flexible than just black box testing.
The extension was developed without changing the original code of Mooshak and RARS simulator.

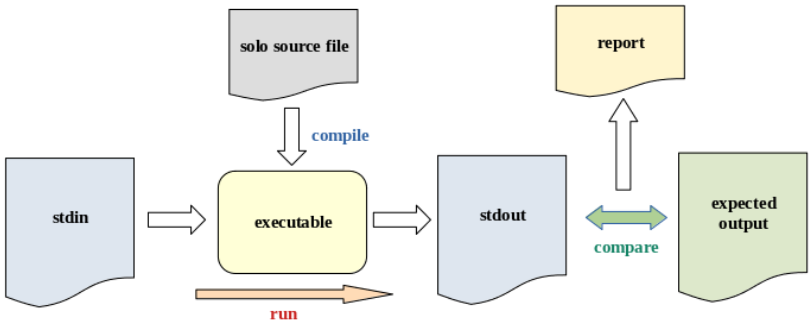

Figure 2: Mooshak default testing operation on solo files

2 Mooshak and RARS simulator testing operation

Mooshak 1.x.x performs black box software testing, running several test cases on the software and generating a report (Daniel [2019]), this is accomplished in the 2 steps defined below, and detailed in figure 2:

1. The submitted **solo source code** file is compiled or assembled first.
2. The application is executed and tested.

To use Mooshak together with RARS simulator, the assembling step is performed by RARS. Also the execution and testing step is performed within RARS simulator environment, being redirected the test cases input and output files to RARS.

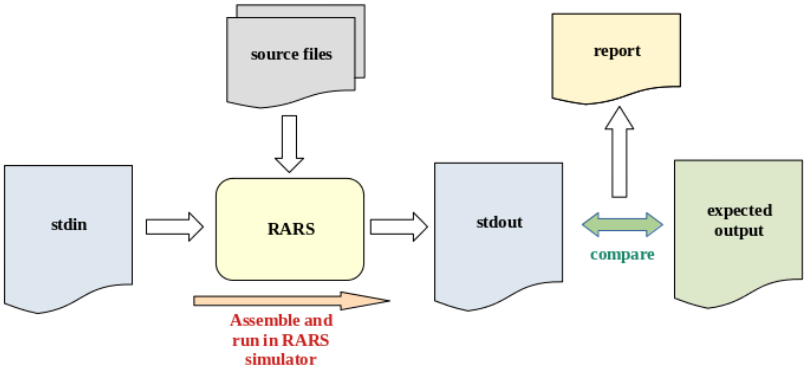

Figure 3: Mooshak and Digital simulator testing operation

3 Extending Mooshak to test RISC-V assembly source code

Extending Mooshak to test multi file assembly source code is done as proposed in (Daniel [2019]), packing all files in a compressed file and submitting this file to Mooshak. The extension unpacks it into a folder before assembling, executing and testing within RARS. This is done at the assembly step, as shown in figure 3, specifying it at language settings page on Mooshak's GUI, **"Compile"** field, see figure 4.

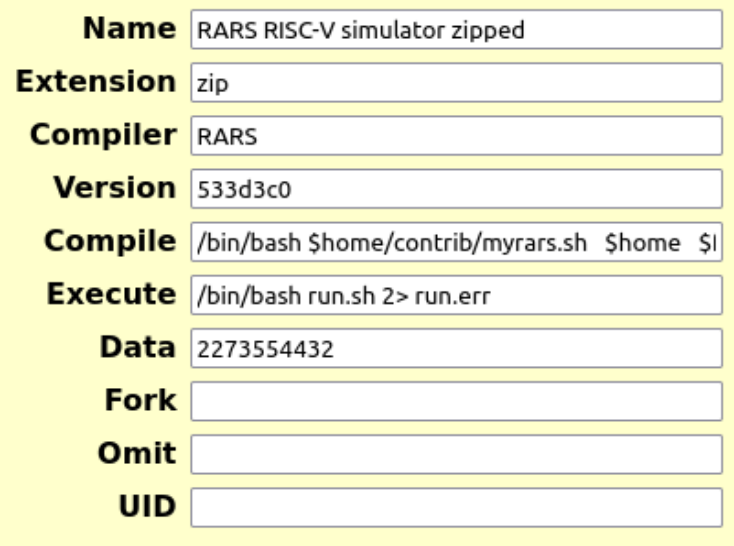

Figure 4: Mooshak language settings

At this field, is called a script passing Mooshak operation variables (Daniel [2019]), that give information on the submission: the home directory where the submitted packed file is stored, its file name and extension.

```
csext.sh $home $file $extension $environment
```

The environment variable allows to define code to be injected along with the submitted code. This gives more flexibility to the testing process, besides just black box testing. This variable can be set in the problem and test setting page of Mooshak, figure 5

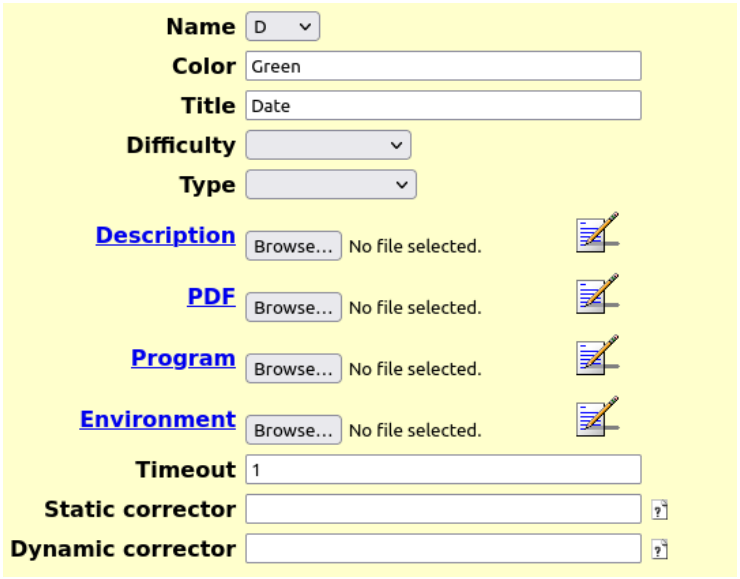

Figure 5: Mooshak problem and test settings

3.1 Execution and testing of RISC-V assembly source code

After unpacking, the assembly of source code is done by RARS. The execution is also done by RARS, in a way that the stdout of RARS is captured and sent to Mooshak to be compared with the expected output. A suitable launching procedure of the application must be performed to accomplish this. This is done by a script, generated to this effect, and called in the **"Execute"** field in the language settings, figure 4.

The main steps of this Mooshak extension are presented below.

Algorithm 1: Extended Mooshak client-server testing (simplified)

$folder \leftarrow unpack(submission)$;
$assembleWithRARS(folder)$;
$generateLaunchScript(folder)$;

RARS is a Java multi threaded application. To execute RARS from Mooshak and support also multi thread applications, it is needed to raise the thread limit in the Mooshak's sandbox **"safe execution environment"**.

This can be configured on the overall language setting, figure 6, fields **MaxCompFork** and **MaxExecFork**. The former specifies the maximum number of threads allowed at Mooshak step 1, assembly, while the latter in Mooshak step

2, the execution and testing.

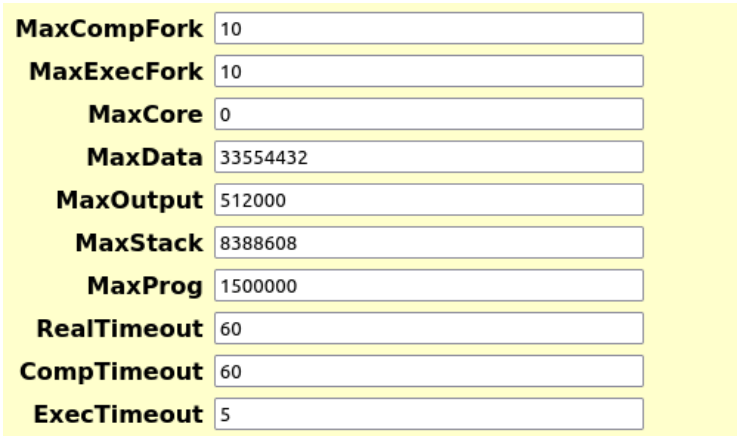

Figure 6: Mooshak all languages settings

This can also be done in Mooshak's language settings page, figure 4, field **Fork**. This field takes precedence on the others.

4 Conclusion

Mooshak allows only the black box testing of solo file programming applications, which is unsuitable for multi file client-server application. However has the ability to receive and perform basic black box testing on source code. This extension adds support for Mooshak to receive multi file source code submissions, compile and execute multi-thread applications, as well as client-server applications. This kind of applications can be run together on the same system to perform black box testing on both. The extension is an add on, that does not change Mooshak original source code, and add the ability to test client-server applications.

References

Helder Daniel. Java unit testing framework, 2019. URL `https://junit.org`. [Online; accessed 7-June-2020].

ICPC. International collegiate programming contest, 2019. URL `https://icpc.global/worldfinals/rules`. [Online; accessed 08-November-2019].

RISC-V international. Full technical specifications, 2021. URL https://riscv.org/technical/specifications/,note="[Online; accessed15-Decembers-2021]".

Benjamin Landers. Risc-v assembler and runtime simulator, 2021. URL https://github.com/TheThirdOne/rars. [Online; accessed 15-Decembers-2021].

José Paulo Leal and Fernando Silva. Mooshak: a web-based multi-site programming contest system. *Software Practice & Experience*, 33(6):567–581, 2003. URL http://www.ncc.up.pt/mooshak/.

Helmut Neeman. A digital logic designer and circuit simulator, 2021. URL https://junit.org. [Online; accessed 23-September-2021].