Automatic Test-Based Assessment of Assembly Programs

Luís Tavares[1] Bruno Lima[1,2] Antonio J. Araújo[1,2]

[1]Faculty of Engineering of the University of Porto
Porto, Portugal

[2]INESC TEC
Porto, Portugal

18th International Conference on Software Technologies
Rome, Italy
July 12, 2023

Context

There has been a growing interest in automatic assessment tools in the last half-century (Douce et al., 2005). This is motivated by:

- Many degrees and educational programs nowadays offer programming courses (Caiza & Álamo Ramiro, 2013);
- The number of students enrolling in these courses is increasing, and professors need means to mass-grade assignments as it appears to be a challenging task (Marchiori, 2022).

Motivation

- Assembly programming is a common subject in Computer Science degrees. However, there is an existing gap in the automatic assessment of assembly exercises;
- Take advantage of our previous work to extend it to a more complete tool;
- The programming environment is not user-friendly, and students need to install a compiler and an emulator to test their programs (DS-5).

Goals

- Use our unit-test assembly grading tool as its foundation to perform unit tests on exercises (Damas et al., 2021);
- System that runs on the browser, easy to use, no installation required;
- Similarity to influential platforms used to learn programming languages (*e.g.*, CodeWars[1], LeetCode[2]);
- Plagiarism detection module to detect similar solutions;
- Open source and modular architecture to support further integrations.

This research presents *AEAS* (ARM [Extensible] Assessment System) to grade ARM64 programming exercises.

[1] www.codewars.com

[2] leetcode.com

Functionalities

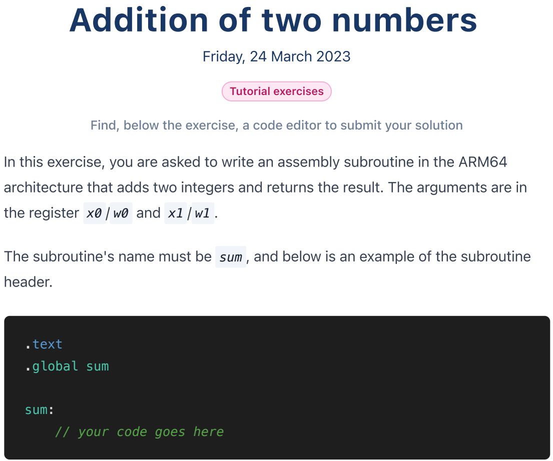
The system provides functionalities for students and professors. However, they are very distinct as their goals are different from one another.

Student functionalities → test exercises, easy to use interaction.

Professor functionalities → grade students' exercises, maintain and manage the system, not so straightforward interaction.

Functionalities

Student functionalities



Addition of two numbers

Friday, 24 March 2023

Tutorial exercises

Find, below the exercise, a code editor to submit your solution

In this exercise, you are asked to write an assembly subroutine in the ARM64 architecture that adds two integers and returns the result. The arguments are in the register `x0` / `w0` and `x1` / `w1` .

The subroutine's name must be `sum` , and below is an example of the subroutine header.

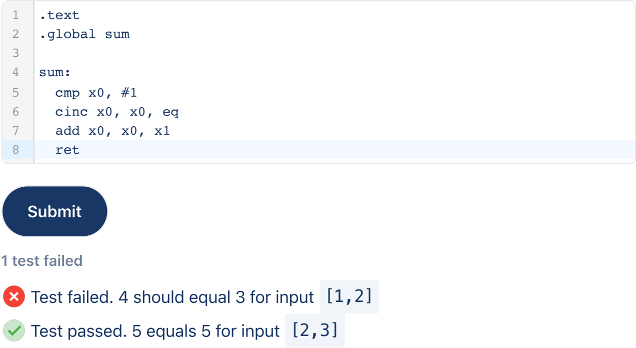
```
.text
.global sum

sum:
    // your code goes here
```

Functionalities

Student functionalities

Students solve the exercise by writing their solution in the editor provided by the system.



```
.text
.global sum

sum:
  cmp x0, #1
  cinc x0, x0, eq
  add x0, x0, x1
  ret
```

1 test failed

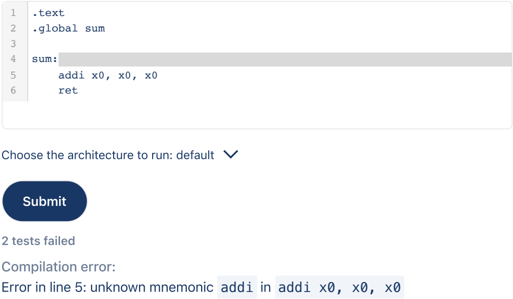
✗ Test failed. 4 should equal 3 for input `[1,2]`

✓ Test passed. 5 equals 5 for input `[2,3]`

Functionalities

Student functionalities

If the code fails to compile, the system will provide the error message returned by the compiler.



```
1  .text
2  .global sum
3
4  sum:
5      addi x0, x0, x0
6      ret
```

Choose the architecture to run: default

2 tests failed

Compilation error:

Error in line 5: unknown mnemonic `addi` in `addi x0, x0, x0`

Functionalities

Professor functionalities

Professors can access a dashboard to manage the exercises and monitor students' progress. The dashboard offers the following functionalities:

- CRUD operations on exercises;
- Grade multiple student submissions on an exercise;
- Track exercise statistics.

Functionalities

Professors functionalities

Professors can perform operations on exercises. The following list defines the properties of an exercise.

- name;
- description (supports $\LaTeX$, Markdown, and code rendering);
- label;
- visibility;
- definition and test cases configuration (via a YAML file, according to the format defined by Damas et al. (Damas et al., 2021)).

Functionalities

Professor functionalities

Professors can grade a set of student submissions using three factors:

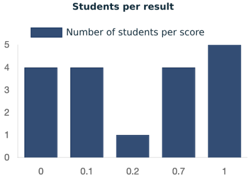
- Unit-tests;
- Instruction presence, whether a code uses a specific instruction or not;
- Plagiarism.

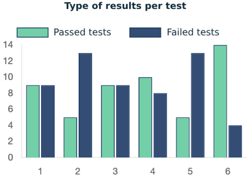
It is possible to run these options individually or combined.

Functionalities

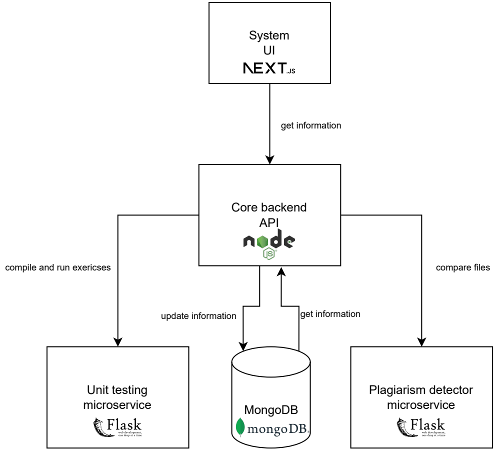
Professor functionalities

Professors can also get an overview of the statistics for exercises.





Architecture



Architecture

Backend

The backend has two submodules, one for running the unit tests using a modified version of the tool created by Damas et al. (Damas et al., 2021), and another for detecting plagiarism.

The unit test runner is responsible for compiling the student's code and running the unit tests provided by the professor.

The plagiarism detector uses Lark, a Python parser library, to parse and compare the student's code with other students' code.

Architecture

Unit-test runner

To perform unit tests on the exercises this work extended[3] our previous work on assembly unit-testing (Damas et al., 2021).

```
sum
  params:
    — int
    — int
  return:
    — int
```

```
sum:
  — inputs: [1, 2]
    outputs: [3]
    weight: 0.5
  — inputs: [2, 3]
    outputs: [5]
    weight: 0.5
```

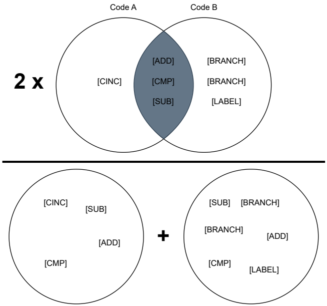
```
{
  "name": "sum",
  "compiled": true,
  "ok": true,
  "passed_count": 2,
  "test_count": 2,
  "score": 1,
  "tests": [
    {
      "weight": 1,
      "run": true,
      "input": [1, 2],
      "output": [3],
      "passed": true
    },
    ...
  ]
}
```

[3] github.com/luist18/areas

Architecture

Plagiarism detector

The plagiarism detector[4] uses context-free grammar to parse the student's code into tokens and compare it with other students' code. The comparison is done using the Sørensen–Dice coefficient.



[4] github.com/luist18/yapy-arm64

How is the grading done?

The final score of a student, i, in an assignment a with a source code s_i is a value between 0 and 1 and is given by the following formulas:

$$score(s_i, a) = \sum_{j=1}^{T} test_a(j, s_i) \times w_a(j) \tag{1}$$

$$test_a(j, s_i) = \begin{cases} 0, & \text{if } s_i \text{ does not pass test } j \\ 1, & \text{if } s_i \text{ passes test } j \end{cases} \tag{2}$$

$w_a(j)$ is the weight of test j for the assignment a. T is the total number of tests for the assignment a.

When the professor specifies an instruction to search for in the source code, the system will search for the instruction in the source code and, if not found, the student's score is 0.

How is the grading done?

Plagiarism makes the process semi-automatic

Why is plagiarism detection missing in the formulas? **Grading is not quite automatic...**

The key message is that plagiarism detection is only an auxiliary tool to narrow the detection of plagiarism cases.

Validation

A survey was conducted on 563 students to evaluate the system. In total, 93 students (16.52%) filled out a questionnaire.

Question 4 On a scale of 1 (very difficult) to 5 (very easy), how would you evaluate the ease of programming in the web-oriented test platform compared to the DS-5 IDE?

1 (DS-5 easier)	**2**	**3**	**4**	**5 (AREAS easier)**
8.5%	6.4%	17.0%	23.4%	44.7%

Table: Results for question 3.

Validation

A survey was conducted on 563 students to evaluate the system. In total, 93 students (16.52%) filled out a questionnaire.

Question 5 On a scale of 1 (only used DS-5) to 5 (only used the web-oriented platform), how do you consider your environment to develop and test the assignments? (Level 3 corresponds to a balanced use.)

1 (Only DS-5)	**2**	**3**	**4**	**5 (Only AREAS)**
4.3%	10.6%	27.7%	31.9%	25.5%

Table: Results for question 5.

Conclusion

- The AEAS tool is a configurable and open-source automatic assessment tool designed for assembly exercises in teaching environments with a large number of students;
- Validation results show that the AEAS tool significantly improved students' understanding of the ARM64 assembly language;
- Future work includes enhanced student features, more professor professors to track students' progress, compatibility with other assembly languages like RISC-V, and integration with other tools.

Questions?

Question time, thank you for your attention!

References I

- Caiza, J. C., & Álamo Ramiro, J. M. d. (2013). Programming assignments automatic grading: Review of tools and implementations.
- Damas, J., Lima, B., & Araujo, A. J. (2021). AOCO - A Tool to Improve the Teaching of the ARM Assembly Language in Higher Education. *2021 30th Annual Conference of the European Association for Education in Electrical and Information Engineering (EAEEIE).*
- Douce, C., Livingstone, D., & Orwell, J. (2005). Automatic test-based assessment of programming: A review. *Journal on Educational Resources in Computing.*
- Marchiori, A. (2022). Labtool: A Command-Line Interface Lab Assistant and Assessment Tool.