# Automated grading system for picoComputer assembly codes integrated within E-Learning platform

Jovan Đukić, Vladimir Jocović, Marko Mišić, *Member, IEEE,* and Milo Tomašević

***Abstract*—Obtaining programming skills is one of the most important prerequisites for a future career of every electrical or software engineer. The programming expertise is best acquired by gradually advancing from simpler to more complex programming paradigms, architectures, and languages. That being the case, a restrictive educational computer architecture – picoComputer, along with a development environment, was developed at the University of Belgrade, School of Electrical Engineering to early expose the students to the concepts of assembly language programming. Having in mind that programming skills are successfully attained only through practical work, such as homework assignments, projects, and laboratory exercises, some more contemporary picoComputer simulation environments were implemented, including MessyLab desktop application and a Picosim web-based solution. However, programming courses at our school are massive and require the utilization of online learning platforms, aiming to properly achieve a scalable learning process. Hence, we employed Moodle E-Learning platform, as well as the CodeRunner plugin, to facilitate and accelerate the teaching and assessing processes in both of our major programming courses. CodeRunner plugin supports various widespread programming languages and is also highly programmable, which is why the integration of picoComputer architecture within a contemporary learning system arose as an opportunity.**

***Index Terms*—E-Learning; automated code assessment; Moodle; picoComputer**

## I. Introduction

Strong programming expertise is one of the fundamental abilities of a contemporary software engineer and a necessary quality of an electrical engineer, as well. Gaining programming practice is essential for not only solving real-world problems in software but also for memory sharpening and achieving an ability to efficiently resolve various problems that are seemingly outside of the programming scope.

Jovan Đukić is with the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: dj@etf.bg.ac.rs)

Vladimir Jocović is with the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: jocke@etf.bg.ac.rs)

Marko Mišić is with the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: marko.misic@etf.bg.ac.rs), (https://orcid.org/0000-0002-7369-4010).

Milo Tomašević is with the School of Electrical Engineering, University of Belgrade, 73 Bulevar kralja Aleksandra, 11020 Belgrade, Serbia (e-mail: mvt@etf.bg.ac.rs)

An effective approach to adopting good programming skills requires a strong theoretical foundation, which is being acquired through traditional methods of lecturing, in addition to a practical approach, which includes homework assignments, larger projects, and laboratory exercises [1].

Programming courses at the School of Electrical Engineering, the University of Belgrade, are mandatory for all first-year students and they are organized into two one-semester courses - Programming 1 and Programming 2. Both courses are mainly focused on studying programming languages (Python and C) and introduce different programming paradigms. They start with a low-level assembly language and continue with a more complex procedural and to some extent object-oriented programming paradigm. Moreover, course topics are permeated with basic data structures and code complexity topics.

Our first-year programming courses are attended by a vast number of students (up to a thousand). These massive courses impose a lot of overhead regarding the process of qualitative assessing the students' work and administering the course contents, as well. Hence, there was a need to establish online learning platforms and other tools intending to ease the whole process of managing these huge courses, as well as disburdening the already overexerted teaching staff. Our first experiences with Moodle E-Learning platform in programming courses are described in [2]. In our previous efforts, we also had a good experience with Moodle e-learning platform for other computer engineering courses [3], as well as with other tools for student assessment [4], analysis of results [1], and source code plagiarism detection [5, 6] which are all widely used in programming education. For all those reasons, we decided to implement appropriate support for the emulation of picoComputer assembly codes in Moodle e-learning platform, as well. We describe our motivation and the details of implementation in the rest of the paper.

The second section expresses the reasons behind the choice to move mandatory programming courses to the e-learning Moodle platform and the course organization within it, as well as the examination process using CodeRunner plugin. The third section presents in more detail the in-house developed architecture for teaching assembly language programming - picoComputer (pC). The fourth section describes the present-day implemented system for compilation, execution, and evaluation of source codes written in the pC assembly language and its integration with the CodeRunner plugin within Moodle platform. The fifth section illustrates the evaluation process and the results obtained by system testing. Finally, a brief conclusion and future work are given in the last section.

 ISBN 978-86-7466-930-3

## II. MOODLE PLATFORM AND CODERUNNER PLUGIN IN PROGRAMMING COURSES

The programming courses lectures at the School of Electrical Engineering are held by professors and teaching associates. The professors mostly teach the theoretical aspects of the currently studied programming topics, giving emphasis to some important essences needed for successful mastering of the practical programming tasks. Bearing that in mind, the teaching associates organize auditory exercises in a more practical manner, thus those classes are dedicated to programming practices only.

Each elementary problem and some intermediate programming tasks are conducted alongside students, aiming to strengthen and solidify their ascending programming skills, as well as to introduce new approaches to solving programming problems. Simple programming tasks were solved using an integrated development environment, such as PyCharm for Python or Microsoft Visual Studio for C, while the more complex ones were worked out on the Moodle platform using the CodeRunner plugin.

Moodle is an online learning platform that allows teachers to create courses for students and to grade their work in those courses using tests. The platform supports a variety of plugins, and we found the CodeRunner plugin the most useful for our grading purposes. This plugin introduces a new type of question, which allows teachers to assess and grade students' source codes. Correctness of the students' codes is partially verified using an automated testing process implemented by the teaching associates, who managed to appropriately configure the plugin using a custom Python script. This feature places the CodeRunner plugin at the top of the list of supported plugins. Unfortunately, some code characteristics still need to be checked manually, e.g., coding style and efficiency.

The example that demonstrates the usage of the Moodle platform is shown in Figure 1. It illustrates the exercise concerning the linked list data structure. The exercise is carried out in C programming language and consists of basic operations performed on linked lists: insertion and deletion of elements, list traverse operations, etc. Before a problem is approached practically, the topic is explained using a PowerPoint presentation. The task itself is straightforward and performed on a simple linked list of integers.

Intending to make the topic of linked lists more interesting, a big task is organized as a series of smaller tasks for a more comprehensive understanding. The task is divided into smaller task sets of varying difficulty. Former task sets consist of commonly used linked list operations and latter task sets functionally depend on the previous task sets. This way the goal is to incrementally build and test the solution and to teach students one of the most important programming principles – code reusability.

As shown in Figure 1, a question has a small table at the top of the page which contains the test input data and the expected output data. Below the test cases table, there is a text area for the code itself. The CodeRunner plugin also includes syntax coloring which is an additional advantage for the students since it can indicate the errors that would be very hard to find in a classical exam notebook.

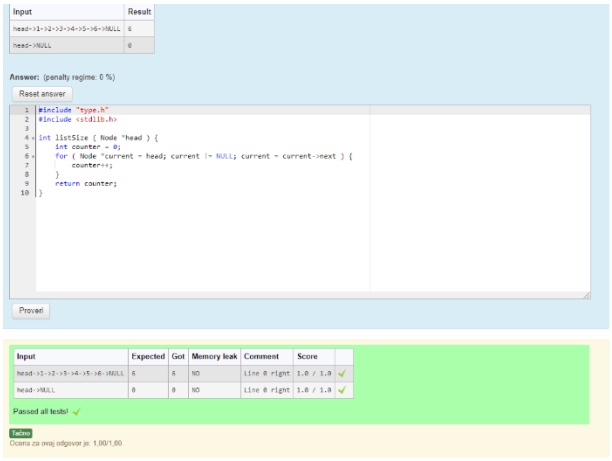


Figure 1 A CodeRunner question in C

After the required piece of code is written in the provided text area, the students can check its correctness by clicking on the check button. When the button is clicked, the contents of the text area are sent to a server dedicated to checking and grading CodeRunner questions. The server first compiles the given code and executes it using the supplied test cases. After execution, the server collects the output and compares it with the expected output.

The main advantage of the CodeRunner plugin for Moodle platform is that the entire process of code checking is configurable [7]. This is achieved through Python scripts which are executed each time a student checks the question. Furthermore, the comparison of the expected and collected output can be graded line by line, thus allowing the students to receive partial points for each successfully passed test case. After the outputs are compared, the result table, shown below the text area in Figure 1, is created giving feedback and their scores to the students, while pointing them to possible errors in the code.

In the last few years, teaching associates were able to master the craft of the plugin configuration and managed to successfully port programming tasks written in Python and C programming languages to the CodeRunner plugin. These high-level programming languages are the foundation of our mandatory first-year courses. Details of the porting process of these high-level languages and further information are presented in [2]. The CodeRunner plugin was successfully used at other universities in other contexts for Python and C++, as well [8].

However, before introducing the students to the concepts of high-level programming languages, they are taught some elementary concepts of low-level assembly programming. Being a relatively minor part of the course, it required an underlying educational architecture that would be quite restrictive, and that's why the aforementioned picoComputer architecture is envisioned and developed. Considering the nature of the low-level languages, the written assembly code can frequently be hard to read, maintain and debug. Even when the source code is syntactically correct, its possible semantic flaws could be tedious to discover. Until recently, teaching associates had to manually inspect the source code, which can devour plenty of time and energy, even for simple programming problems.

Taking these circumstances into account and having a positive experience gained from porting high-level

 ISBN 978-86-7466-930-3

programming languages such as Python and C to Moodle using the CodeRunner plugin, teaching associates decided to establish a system that could verify and test the students' solutions written in the picoComputer assembly language on the e-learning platform. Nevertheless, the task of implementing such a system is inherently harder than above mentioned porting challenges. Teaching associates had to provide not only a configured environment for executing students' source codes using test examples but also an implementation of a compiler and an emulator for a source code written in assembly language was necessary. These components were not needed in previous porting undertakings, since there are numerous compilers and interpreters for widespread programming languages, including C and Python.

## III. PICOCOMPUTER

In 1989. Prof. Jozo Dujmović designed a computer architecture named picoComputer (pC) [9] and developed a DOS application *pC Assembler and Simulator* (pCAS). His intention was to facilitate the teaching and understanding of the assembly languages, which are naturally, due to their low-level nature, to some degree demanding. Since only a part of the introductory programming course is devoted to low-level programming, the pC is designed as a quite restrictive architecture as implied by its name. However, even with its restrictive scope, pC is still very useful. In order to provide more convenient environments, two tools have been recently developed at our school, MessyLab IDE [10], and a web online environment called picoSim [11].

Although the picoComputer architecture is more than 30 years old, it is still relevant nowadays. Its aim is to provide a framework for demonstrating assembly-level programming. It follows the classical Von Neumann architecture and, even though the characteristics of various components have changed throughout time and the instruction sets are getting more and more complex, the basic principles of computer structure have not changed a lot. The picoComputer generally consists of the Central Processing Unit, Random Access Memory, and Input/Output devices, which are all connected using a shared Bus, as shown in Figure 2.

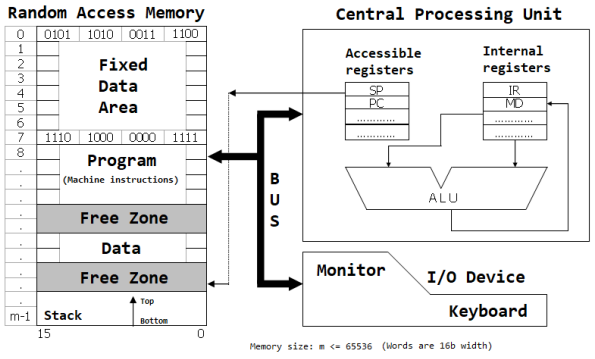


Figure 2 The picoComputer architecture

The Central Processing Unit has several internal registers, yet none of them are directly accessible as the instruction operands. Because of a limited instruction format, there are no general-purpose registers. Still, Program Counter (PC), which points to the next instruction to be executed, and Stack Pointer (SP), which indicates the top of the stack, are registers that can be indirectly manipulated by certain instructions. The value of the PC register is either incremented after an executed instruction or can be directly loaded with the branch address by a control instruction. SP register value is affected by subroutine handling instructions and.

The Random Access Memory consists of 65536 locations (memory words), which means that memory addresses are 16 bits wide, while each location is, also, 16 bits wide. The memory is logically divided into two sections: Fixed Data Area and Free Area. Fixed Data Area includes the first 8 locations, which are directly accessible through direct memory addressing. Free Zone is comprised of the remaining locations and these locations are only accessible indirectly, through another location from the Fixed Data Area, using memory indirect addressing. These locations can be used arbitrarily. The third addressing mode is the immediate addressing where the operand is found in the instruction itself.

The input device is a keyboard, and the output device is the monitor. Numerical data can be entered using a keyboard, while the screen presents the contents of certain memory locations. The input/output operations are blocking operations. Consequently, there is no need for polling a status register. However, parallelism is not supported.

Every picoComputer program consists of two sections: the directive section, and the instructions section. There are two kinds of directives: symbol definition directives and the origin directive. A symbol definition directive is used to assign numerical values to symbols to improve code readability. These symbols are replaced by their numerical values in the assembling process. Labels are an implicit means of symbol definitions, and they can be specified by an identifier attached to any instruction. The origin directive defines the starting memory location where the executable code (instruction section containing instructions following the origin directive) resides.

Every instruction is defined by its symbolic mnemonic and a variable-length comma-separated list of operands. The picoComputer format provides up to 3 operands in an arbitrary instruction. Instructions can occupy one or two memory words (16 or 32 bits). The operation code and three operand fields are encoded in four 4-bit nibbles of the first word, as shown in Figure 3.

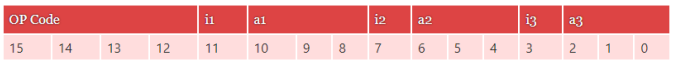


Figure 3 Typical picoComputer one-word instruction format

Hence, a maximum of 16 different operation codes are supported. Each operand specification consists of *i* field (1 bit) and *a* field (3 bits), where *i* indicates the memory addressing mode (0 for direct or 1 for indirect), while *a* field represents an address from Fixed Zone Area (0-7). The 16-bit immediate operand, when supplied, is stored in the second instruction word. The pC instruction set consists of the integer arithmetic instructions (addition, subtraction, multiplication, division), data transfer instruction (scalar or

 ISBN 978-86-7466-930-3

vector move), conditional branches, subroutine call and return, I/O instructions, and stop instruction.

## IV. ASSESSMENT OF STUDENTS' PICOCOMPUTER ASSEMBLY PROGRAMS

As stated, in the second chapter, CodeRunner allows its users to provide a custom Python script for the purpose of grading and assessing. This script is executed in a preconfigured CodeRunner plugin environment, which provides numerous predefined variables. These variables can be used to obtain a variety of information about students from Moodle (i.e., student profile information) and, more importantly, the answer submitted by the student through the CodeRunner form. Given that the pC is our custom assembly language and that the answer is given in a text form, the authors had to build a custom compiler and an emulator to be able to grade and assess students' answers.

The compilation phase consists of text manipulation and performs syntax and semantic checks specified by the rules of the assembly language. During this stage, the text is split into individual lines, which are checked separately. If the process is successful, the result is a Python list data structure of integers, where each element represents an individual memory location. However, if there is an error in the code, the result of the compilation phase is a list containing descriptions of each individual erroneous line (line number and the error description). The unsuccessful compilation phase is depicted in Figure 4.

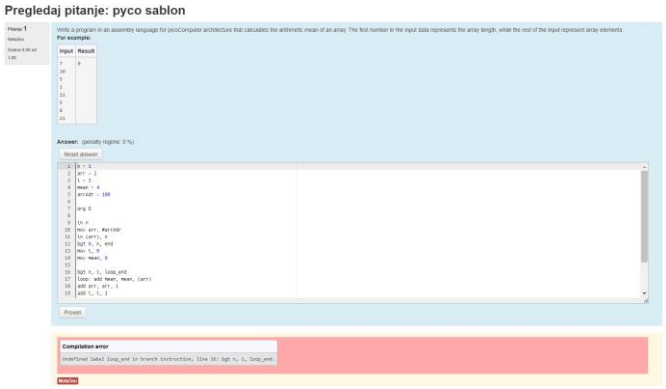


Figure 4 Unsuccessful pC compilation phase

If the compilation phase is successful, the following step is the emulation phase. The input data for the emulation phase is also a list of integers (i.e., memory locations), the address of the first instruction provided by the origin directive and a list of integers representing the input data. The emulation phase is a simple *for*-loop, which reads instructions one by one, executes them and stores their results in the memory. The exceptions to this workflow are the IN and OUT instructions. The IN instruction reads one or more numbers from the input data list, while the OUT instruction writes the content of one or more memory locations to the output list. This output list is later used for comparison with the expected results. Runtime checks are also performed during this stage. Given the restricted nature of the pC and the fact that only integer data type is supported, the only runtime check performed is the division-by-zero check. This is depicted in Figure 5.

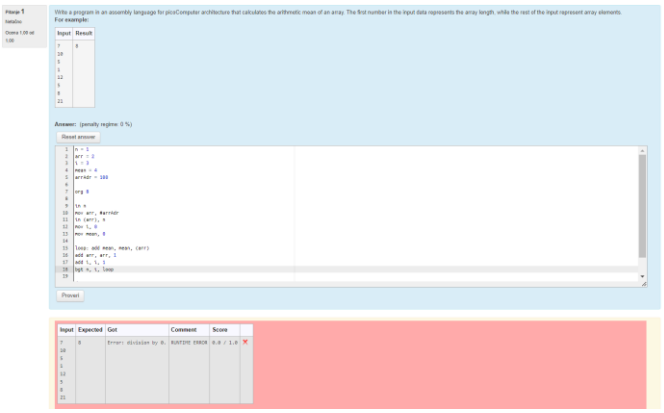

Figure 5 Runtime error checks performed

If both the compilation and the emulation phases are successful, the result of the emulation phase is compared to the expected result and the final grade is formed. The comparison is performed on a line-by-line basis, where each line gives the same number of points. Finally, the student is presented with the score table as shown in Figure 6.

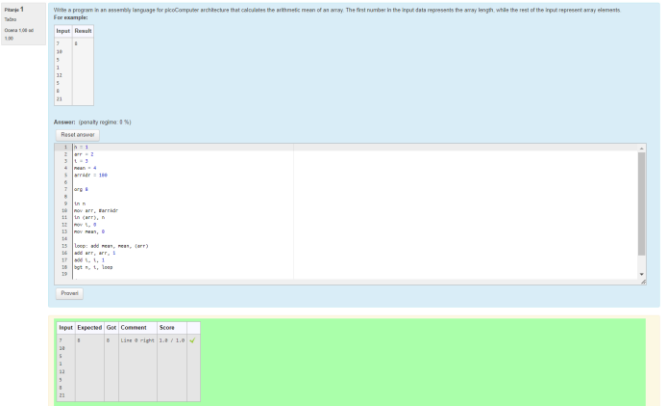

Figure 6 Score table for a program

## V. SYSTEM TESTING AND EVALUATION PROCESS

During the development, the system was tested with custom-written programs, which cover all possible valid and invalid instruction formats. Valid programs consist of instructions as specified by the pC instruction format. These programs are relatively easy to write since there is a limited number of possibilities that are in accordance with the syntax and semantic rules. However, the number of invalid formats is far greater in number and, therefore, it is impossible to cover all of them. Hence, the system was also tested with the source codes of the students in the previous years.

Until recently, during exams students used the MessyLab desktop application to write and self-check their solutions. Regardless, the students used the Moodle platform to submit their answers, which were graded manually. We used these answers to perform evaluation by comparing manual scores given by the teaching associates and the scores given by the system. The system was evaluated on students' programs after the exams have passed and the results of both approaches were quite similar. We sincerely hope that the live results will be as good as these.

 ISBN 978-86-7466-930-3

## VI. CONCLUSION

In this paper, we have presented the migration process of a restrictive educational picoComputer architecture to the online e-learning Moodle platform using a dedicated CodeRunner plugin for our introductory programming course. With this new feature, we have successfully fully moved all our programming activities within the Programming 1 course to the LMS. Moodle and CodeRunner are extensively used during auditory exercises in the computer lab and examination. The students reacted very positively to this innovative activity.

Having migrated all programming languages in our mandatory programming courses, it may seem that the future work lacks required matter to be considered worthy. However, the is a significant potential concerning the available possibilities to parametrize and configure these types of systems not only to achieve more sophisticated means of grading, yet also to broaden the spectrum of conceivable programming task types. In the future, we have an intention to develop various learning activities in Moodle, as well to extend our coding exercises pool with more assignments written in different programming languages. Moreover, we have in mind to migrate the rest of our programming courses to such type of learning and examining.

## ACKNOWLEDGMENT

This work was supported by the Science Fund of the Republic of Serbia, grant no. 6526093, AI-AVANTES, as well as the Ministry of Education, Science, and Technological Development of the Republic of Serbia (III44009 and TR32047). The authors gratefully acknowledge the financial support.

## REFERENCES

[1] M. Mišić, M. Lazić, and J. Protić, "A software tool that helps teachers in handling, processing and understanding the results of massive exams," in *Proceedings of the Fifth Balkan Conference in Informatics*, 2012: ACM, pp. 259-262.

[2] V. Jocović, J. Đukić, and M. Mišić, "First Experiences with Moodle and Coderunner Platforms in Programming Course," in *Proceedings of the Tenth International Conference on e-Learning*, Belgrade Metropolitan University, Belgrade, 2019, pp. 81-86.

[3] D. Drašković, M. Mišić, and Ž. Stanisavljević, "Transition from traditional to LMS supported examining: A case study in computer engineering," *Computer Applications in Engineering Education*, 2016.

[4] A. Bošnjaković, J. Protić, D. Bojić, and I. Tartalja, "Automating the Knowledge Assessment Workflow for Large Student Groups: A Development Experience," *International Journal of Engineering Education*, vol. 31, no. 4, pp. 1058-1070, 2015 2015.

[5] M. Mišić, Ž. Šuštran, and J. Protić, "A Comparison of Software Tools for Plagiarism Detection in Programming Assignments," *International Journal of Engineering Education*, Article vol. 32, no. 2, pp. 738-748, 2016 2016.

[6] M. J. Mišić, J. Ž. Protić, and M. V. Tomašević, "Improving source code plagiarism detection: Lessons learned," in 2017 *25th Telecommunication Forum (TELFOR)*, 2017: IEEE, pp. 1-8.

[7] R. Lobb and J. Harlow, "Coderunner: A tool for assessing computer programming skills," *ACM Inroads*, vol. 7, no. 1, pp. 47-51, 2016.

[8] D. Croft and M. England, "Computing with CodeRunner at Coventry University: Automated summative assessment of Python and C++ code," in *Proceedings of the 4th Conference on Computing Education Practice 2020*, 2020, pp. 1-4.

[9] J. J. Dujmović, Programski jezici i metode programiranja – odabrana poglavlja. *Akademska misao*, 2004.

[10] M. Anđelković. "MessyLab project." http://messylab.com/ (accessed April 2022).

[11] N. Miljković. "picoSim project." https://picosim.app/ (accessed April, 2022).