Ferramenta para o auxilio de aprendizado de programação de baixo nível

Igor Fagundes Rezende Silva[1] **Carlos Augusto Paiva Da Silva Martins (orientador)**[1]

[1]ICEI - PUC Minas

Abstract. *The modern world inherently relies on the ability of computational devices to perform data analysis and processing. Despite recent advancements in the computational field [Mack 2011], it is evident that this demand only continues to grow over the years. Coupled with the significant shortage of development professionals expected in the next decade [Group 2023], the following scenario arises: a humanity that depends, in all spheres, on computing and will inevitably suffer from computational starvation. In this sense, one way to combat this issue is to train individuals who possess programming skills and, above all, have an intimate knowledge of their working tool. As a means to address this challenge, this project proposes a platform to aid the learning of low-level programming.*

Resumo. *O mundo moderno depende de maneira intrínseca da habilidade dos dispositivos computacionais de performar análise e processamento de dados. Mesmo com os recentes avanços na área computacional [Mack 2011], é notório que esta demanda só cresce ao longo dos anos. Somado ao fato que haverá significativa falta de profissionais de desenvolvimento na próxima década [Group 2023], cria-se o seguinte cenário: uma humanidade que depende, em todas as esferas, da computação, e que vai, quase que inevitavelmente, sofrer de inanição computacional. Neste sentido, uma maneira de combater essa mazela passa a ser formar indivíduos que tenham habilidade na programação de computadores, e que sobretudo, conhecem intimamente o seu instrumento de trabalho. E como instrumento para este combate, este trabalho propõe uma plataforma de auxílio ao aprendizado de programação de baixo nível.*

1. Introdução

A eficiência da humanidade em gerir e processar dados, controlar sistemas e coisas semelhantes depende intimamente da nossa capacidade de programar sistemas computacionais, capacidade esta que deve não somente abrigar habilidades em eficiência na codificação e organização de código, mas principalmente capacidade de gerir com mestria os recursos da máquina. Quanto melhor um programador consegue gerir os recursos da sua ferramenta de trabalho, melhor o software irá performar e menor será a quantidade de recursos computacionais requeridos para a execução, e por consequência menor será o custo teórico da operação. Assim como em muitos modelos que tentam prever a fome existe a razão entre a capacidade produção calórica da indústria de alimentos versus a demanda calórica de um grupo de pessoas, devemos pensar na mesma razão aplicada ao contexto computacional: qual são os recursos computacionais disponíveis pela humanidade versus aquilo que a mesma demanda. Tal número talvez seja difícil de obter, mas a lógica permanece: quanto menor a demanda de recursos dos softwares individualmente, menor será a chance da raça humana sofrer com a abstinência de recursos computacionais.

Nesse contexto, fica evidente a necessidade de formar profissionais que tenham habilidade na programação de baixo nível, uma vez que, esta modalidade de programação força os desenvolvedores a gerir com eficiência os recursos da máquina. E mais do que isso: saber programação de baixo nível pode garantir ao programador a ciência de como os mecanismos da programação de alto nível ocorrem, e com isso melhorar a eficiência do software de alto nível.

Não haveria tanta urgência no tema do baixo nível se uma boa parcela dos recém formados programadores tivessem habilidade no tema. Infelizmente, maior parte do uso pelos programadores [Overflow 2022] e do foco das instituições de aprendizado é na programação de alto nível [Gallego-Durán et al. 2021], o que pode gerar, e gera, programadores que possuem pouca habilidade de criar softwares capazes de consumir recursos computacionais de maneira eficiente. Uma defesa às instituições seria: "não é mais fácil e lógico focar e ensinar o alto nível primeiro, e quando o programador tiver certa maestria focar no baixo nível?". Uma resposta lógica seria: "O que é mais fácil: ensinar ao indivíduo diversos conceitos lógicos e abstratos de programação, como loops e chamadas de funções, ou começar pela simples realidade que um computador realiza por baixo dos panos simples operações, que mais se parecem com uma calculadora de mercearia?". Logo, além de necessário, o aprendizado de programação de baixo nível em iniciantes, é algo lógico. Diante disso surge então o desafio de como promover de maneira saudável e eficiente este aprendizado.

O objetivo deste trabalho é em averiguar a possibilidade de se utilizar uma plataforma virtual de aprendizado, que seja acessível, intuitiva, com boa capacidade de expansão, e que possua uma estratégia bottom up de ensino. A ideia é ser uma ferramenta prática, capaz de ser útil no aprendizado de programação em âmbito geral, mas sem deixar de maneira alguma de abordar conceitos que existem na programação de baixo nível [Pegoraro and Franchin 2022], e que permita uma boa interação entre o tutor e o aluno.

Assim como em muitas linguagens de programação onde a qualidade é definida mais pelo conjunto da obra do que em alguma feature revolucionária, exemplo de Golang e Zig. Fica claro, que neste software deve seguir este mesmo modelo. É menos sobre alguma feature revolucionária e mais sobre um caminho, uma meta a ser seguida, sobre tomar decisões que tendem a não desviar desta rota. Este problema é muito complexo para existir uma decisão ideal e absoluta no tocante à arquitetura deste software proposto. Se assim não fosse, a maneira de desenvolver software não mudaria tanto ao longo dos anos. Logo, muitas das decisões tomadas no trabalho são frutos de uma análise do que seria o ideal para se manter neste caminho, e portanto, são escolhas.

O software segue um modelo de sandbox, onde o tutor cria uma arquitetura virtual e aluno interage com essa arquitetura por meios dos módulos visuais que o tutor criar. O aluno consegue visualizar, depurar e programar para esta arquitetura virtual. Tudo isso ocorre em uma espécie de *canvas* (tela virtual onde estão dispostos os elementos): o aluno consegue chamar módulos visuais para interagir com a arquitetura que o tutor criou. E qualquer destes módulos visuais, que são plugins, é possível colocar um editor de linguagem de montagem para a arquitetura virtual. O objetivo é criar um ambiente que seja flexível e ao mesmo tempo interessante [da Silva et al. 2018]. Para criar a arquitetura e os módulos visuais é utilizada a linguagem Lua.

2. Trabalhos relacionados

2.1. Um Modelo para Promover o Engajamento Estudantil no Aprendizado de Programação Utilizando Gamification

O artigo [da Silva et al. 2018] discorre acerca do uso das afeições presentes em jogos como uma maneira de engajar o processo de aprendizagem, o artigo apresenta resultados de teste feitos com um comunidade de alunos.

2.2. Um simulador de apoio ao ensino de linguagem de montagem

O artigo [Pegoraro and Franchin 2022] mostra a criação de uma plataforma com o intuito de auxiliar no aprendizado de programação de arquitetura de computadores, mais especificamente linguagens de montagem.

2.3. Low Level Programming Learning Support System by Implementation of Virtual Machine

O trabalho [Katagai and Fukuda 2017] explora a possibilidade do aprendizado de baixo nível por meio da criação de uma máquina virtual, função esta que é semelhante ao papel do tutor no nosso trabalho, logo, é possível tem uma experiência de aprendizado significativa nos dois papeis propostos para o nosso software.

2.4. A low-level approach to improve programming learning

O trabalho [Gallego-Durán et al. 2021] possui uma abordagem extremamente semelhante a este artigo no tocante ao reconhecimento das vantagens do foco no ensino da programação de baixo nível em detrimento do de alto nível em iniciantes.

2.5. SMAC-based Programming Learning Tool: Validating a Novel System Architecture

O trabalho [Rahim et al. 2022] discorre acerca do aprendizado colaborativo no processo de aprendizagem de programação, o que reforça a significância da arquitetura tutor aluno presente no no software desenvolvido neste artigo.

2.6. A teaching approach for bridging the gap between low-level and high-level programming using assembly language learning for small microcontrollers

O trabalho [Bolanakis et al. 2011] fala acerca da possibilidade de utilizar micro controladores para o aprendizado de linguagem de montagem ao invés dos ambientes modernos, que muitas vezes são hostis a quem deseja aprender e executar programas escritos diretamente em linguagem de montagem.

2.7. Learning Computer Architecture with Raspberry Pi

O livro [Eben Upton 2016] fala acerca da possibilidade de utilizar o computador Raspberry Pi como uma ferramenta para o aprendizado de arquitetura de computadores. O livro relaciona-se com este artigo pelo fato de tentar simplificar o ensino de arquitetura de computadores.

3. Metodologia

Como a proposta é criar um ambiente flexível para o aprendizado de programação por meio da arquitetura de computadores, seria lógico que no software existisse uma interface para a criação de módulos e plugins, que podem ser ferramentas e partes de arquiteturas. Com isso em mente, o programa adquire fortes qualidades de uma sandbox, mais do que uma aplicação que fornece uma arquitetura específica, logo, é de extrema importância a criação de um ambiente coeso e harmônico em que os módulos possam existir e coexistir. A arquitetura escolhida desse software é baseada no modelo onde os tutores, primariamente, criam os módulos, sendo que estes podem ser ferramentas ou chips. O aluno também pode estudar por conta própria utilizando exemplos previamente fornecidos.

O núcleo da aplicação foi escrito na linguagem de programação Odin, que é compilada e diferentemente da linguagem C, possui bibliotecas para abertura de janelas e renderização incluidas no compilador, além de diversas adições na sintaxe e semântica que proporcionam melhor produtividade e menor taxa de ocorrência de erros. Para a parte gráfica da aplicação foi utilizada, até o momento, a biblioteca Raylib, executando sobre uma biblioteca de modo imediato criada por mim. Não foi utilizada alguma biblioteca de UI pronta, uma vez que, das bibliotecas de UI de modo imediato que existiam, nenhuma atendeu os requisitos de flexibilidade e simplicidade que era preciso. Foi escolhido o modo imediato para a UI, uma vez que é mais simples de desenvolver usando elas, e, acredito, que é mais simples para a criação das ferramentas.

A arquitetura escolhida é modular, baseada em plugins, para garantir mais flexibilidade no processo de aprendizagem. Os módulos devem ser criados na linguagem Lua, foi escolhida a linguagem Lua pelo fato dela ser bem popular na comunidade das linguagens de script, e pelo fato dela possuir uma simples API para a linguagem C (ou qualquer linguagem que seja capaz de usar a FFI de C), algo que Javascript, por exemplo, não possui.

Para o tutor não precisar criar todas as interfaces de interação com o usuário do ”absoluto zero”é fornecido para ele, e o mesmo consegue habilitar isso por meio da API de Lua, um editor de linguagem assembly integrado. Esse editor não se comporta como um editor de texto comum, e se assemelha mais à um editor de AST (Árvore de Sintaxe Abstrata), onde o aluno pode desviar pouco do padrão de codificação, o que facilita o processo de codificação e compilação.

Os alunos, irão consumir estes módulos e montar dois ambientes, o ambiente da arquitetura, onde irão usufruir dos chips criados pelos tutores; e o ambiente da workspace, onde irão usufruir das ferramentas criadas pelos tutores. Estas ferramentas podem ser diversas coisas, como: editores, debuggers, visualizadores ou qualquer coisa que o tutor desejar criar. O workspace funciona como uma espécie de *canvas* (tela virtual onde estão dispostos os elementos), de área de trabalho: o aluno instancia os modulos para um ”quadro branco”e monta os modulos da maneira que desejar no seu quadro. E é por meio desta workspace que o aluno vai interagir com a arquitetura. É importante frisar que o aluno pode, e será incentivado, a criar seus próprios módulos, e utilizar sua criatividade não somente para usar os módulos já criados, podendo o aluno, gradativamente transicionar do estado de aluno para o estado de tutor.

Outro aspecto importante que é explicar como funciona, do por ponto de vista

da arquitetura da aplicação, os chips. Cada chip possui um nome, também o mesmo possuir um array de memórias, e cada memória tem o seu tamanho e um vetor de labels, o chip também pode requisitar o framebuffer de vídeo, e especificar o a altura e largura em pixels, o chip também especifica como será feita a montagem de assembly para o chip específico, e também o procedimento de ”tick”. As labels para uma memória funcionam como etiquetas que as ferramentas podem ler para exibir com mais precisão o conteúdo das memórias, se por exemplo, um tutor criou um visualizador de registradores à partir das labels de uma memória, a ferramenta irá funcionar mesmo que a arquitetura mude. A label contém o nome, o endereço e o tamanho. Um bom exemplo é um banco de registradores com 4 registradores de 8 bits: para declarar um chip com esta configuração basta criar uma memória com 4 bytes, e 4 labels. Se nesta arquitetura existerem os registradores ”pc”, ”acc”, ”x”e ”y”. As labels serão como: ” name = ’pc’, addr = 0, size = 1 , name = ’acc’, addr = 1, size = 1 , name = ’x’, addr = 2, size = 1 , name = ’y’, addr = 3, size = 1 ”. O procedimento de montagem recebe como input a linha assembly, já fragmentada em partes e cada parte já extraída seu significado útil, e retorna o bytecode que será lido pelo chip, referente ao input passado. Ao ser chamado o procedimento de montagem, a aplicação irá chamar esse procedimento varias vezes, para assim compor as instruções que serão lidos pelo procedimento de ”tick”. O procedimento de ”tick”é onde a lógica do chip irá de fato acontecer, neste procedimento você pode acessar as memórias, a memória de instruções e o framebuffer.

As ferramentas interagem com os chips por meio de uma interface de descobrimento (ainda não implementada), em que podem pesquisar os chips que estão presentes na arquitetura atual. No futuro os chips vão possuir, tags, como: ”CPU, GPU, RAM, ...”, assim como as memórias: ”REGISTERBANK, RAM, TLB, ...”, assim as ferramentas poderão ser precisas no que interagir ou mostrar. Por exemplo: um visualizador de registradores pode focar em só mostrar memórias com a tag ”REGISTERBANK”. Cada memória ou chip possui um id, e por meio deste id é possível realizar diferentes tipos de operações, como escrever ou ler uma memória de um chip, ler as labels de uma memória, alterar ou ler instruções, realizar o build de um conteúdo de um editor para a memória de instruções de um determinado chip, chamar a função de tick do chip, entre outras operações.

Listing 1. Codigo em lua exemplificando uma ferramenta

```
Manifest = {
  name = "Registradores",
  kind = "tool",
  description = [[
  Este  um editor de teste
  ]],
}

ToolUpdate = function(x, y, w, h)
  local info, labels = arch.get_memory_info(cpu_id,
     registers_bank_id)
  local viewer_y = y

  for i, label in pairs(labels) do
```

```
    ui.label(tostring(i), x, viewer_y, w / 3, 20)

    ui.label(label.name, x + w / 3, viewer_y, w / 3, 20)

    ui.label(tostring(arch.get_memory_byte(cpu_id,
       registers_bank_id, label.addr)), x + w / 3 * 2,
       viewer_y, w / 3, 20)

    viewer_y = viewer_y + 20
  end
end
```

Listing 2. Codigo em lua exemplificando um módulo de arquitetura

```
Manifest = {
  name = "Teste CPU",
  kind = "chip",

  description = [[
  Esta eh uma CPU de teste.
  ]],

  memories = {
    {
      size = 4,
      labels = {
        { name = "pc" , addr = 0, size = 1 },
        { name = "acc", addr = 1, size = 1 },
        { name = "x" , addr = 2, size = 1 },
        { name = "y" , addr = 3, size = 1 },
      },
    },
  },
  programmable = true,
  request_framebuffer = true,
  framebuffer = {
    width = 256,
    height = 256,
  },
}

CompileAsmLine = function(mnemonic, param1, param2, param3,
   writer, errors)
  ...
end

ChipTick = function(resources)
  ...
end
```

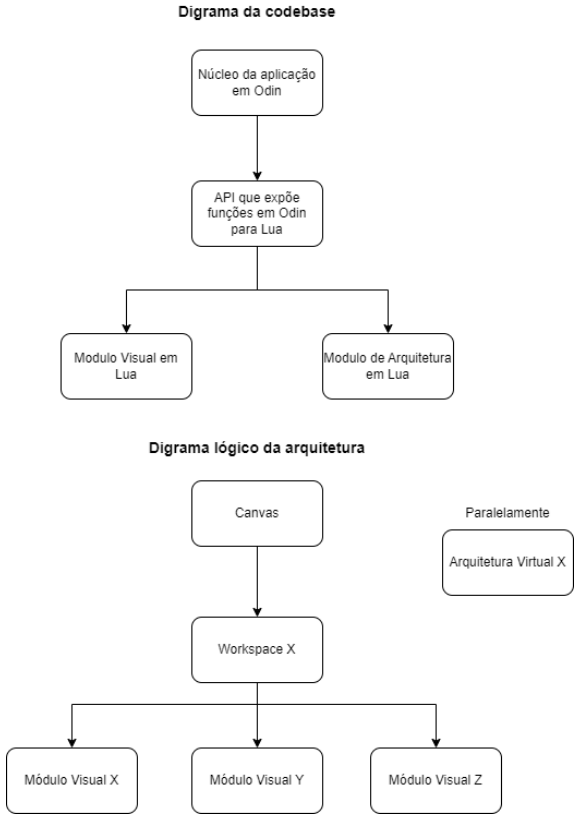

Figura 1. Esquematização básica da arquitetura

4. Resultados

O aplicação final é distribuida na forma de um arquivo executável, tanto Windows quanto Linux. Basta somente o usuário executar este arquivo, sem a necessidade de instalação. O procedimento de usagem da aplicação é simples, e divido em 4 áreas: ”importação”, ”instânciação”, ”arquitetura”e as workspaces. Na aba de importação, o usuário importa novos módulos Lua ao sistema. Na ”instânciação”o usuário consegue colocar um módulo visual na sua área de trabalho. Na aba ”arquitetura”é possível mudar a arquitetura atual. E por último, é possível visualizar as workspaces.

Durante o processo diversas decisões se mostraram boas, outras nem tanto. No tocante ao desenvolvimento da aplicação utilizar um editor de nodos ao invés do tradicional editor de texto se mostrou um empecilho na flexibilidade da sintaxe da linguagem de montagem que é usada pelo aluno. Como o editor de nodos facilita a codificação de um tipo específico de linguagem de montagem, bem MIPS-like, é difícil fazer algo muito diferente dessa estrutura. O editor de nodos é melhor falado na seção de resultados. Outra decisão que não se mostrou muito feliz foi estruturar a API para a criação dos módulos e arquiteturas por meio de chamadas de funções de leitura e escrita, ao invés de expor para a linguagem de script uma estrutura de dados. Tal abordagem se mostrou um problema performático e acabou gerando uma maior complexidade na criação da API. Uma outra escolha que se apresentou como não ideal, mas foi corrigida no meio do processo, foi expor uma biblioteca de interface gráfica de modo imediato para a confecção dos módulos visuais. O problema desta escolha pode ser sintetizado em duas razões: a primeira é que a biblioteca não estava madura o suficiente e tinha problemas do sistema de layout, logo

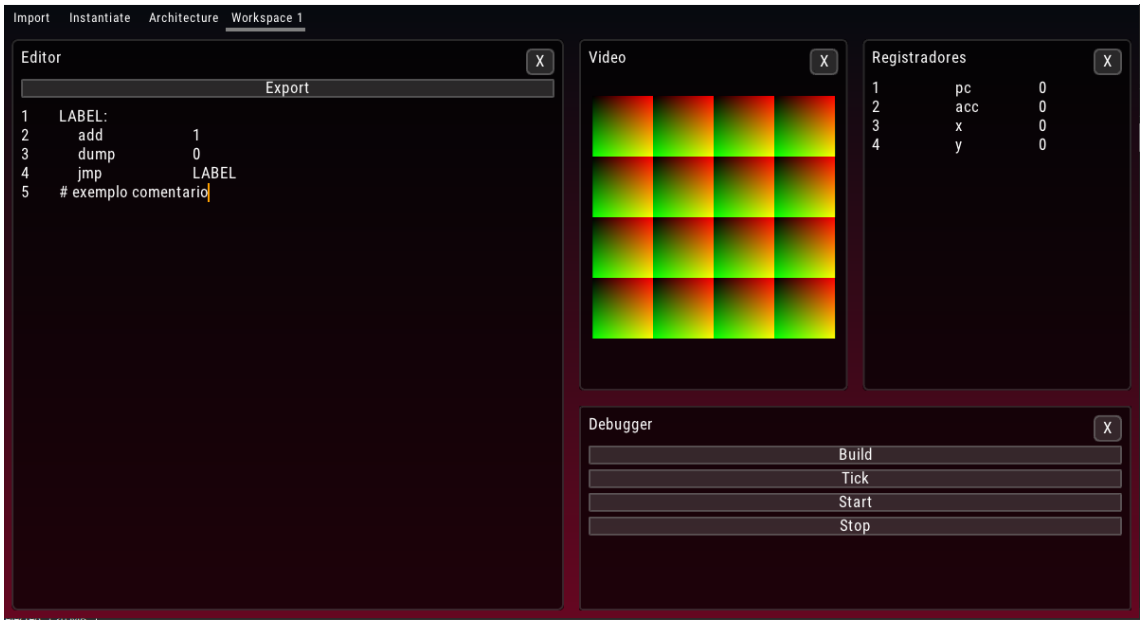

Figura 2. Captura de tela da aplicação, mostrando um *canvas* (tela virtual onde estão dispostos os elementos) preenchido com um editor de texto, um visualizador de vídeo, um visualizador de registradores, e um controlador de depuração, todos estes módulos presentes no *canvas* (tela virtual onde estão dispostos os elementos) podem ser editados pelo usuário por meio da linguagem lua

era difícil posicionar os elementos de UI dentro do módulo visual, o segundo é que é importante dar a flexibilidade ao tutor de poder posicionar manualmente os elementos na tela. A solução foi um meio termo: dar um bypass no sistema de layout da biblioteca e deixar que o script do módulo do tutor posicione manualmente os elementos de UI. E por último, uma decisão que necessita de melhor revisão em um trabalho futuro é o uso de uma linguagem interpretada como o método para o desenvolvimento da arquitetura e dos módulos visuais., nem toda linguagem de script precisa ser interpretada. O problema é que a velocidade de execução ficou distante perto de testes feitos de maneira compilada e nativa. Enquanto em testes compilados era possível interagir com todos os pixels do vídeo 470 por 280 da arquitetura virtual 60 vezes por segundo, testes com a arquitetura definida na linguagem Lua demoram consistentemente dezenas de segundos para interagir com um vídeo 256 por 256 pixels.

É importante ressaltar também que muitas decisões se mostraram benéficas ao longo do processo. O uso de um sistema de módulos possui grande potencial de agregar extrema flexibilidade no processo de aprendizagem, uma vez que o tutor pode simular qualquer arquitetura teórica. O sistema de *canvas* (tela virtual onde estão dispostos os elementos) também se mostrou uma boa decisão, foi possível criar uma arquitetura onde a disposição dos módulos visuais fica completamente a cargo do aluno, podendo o mesmo configurar o seu *canvas* (tela virtual onde estão dispostos os elementos) para diferentes situações e necessidades.

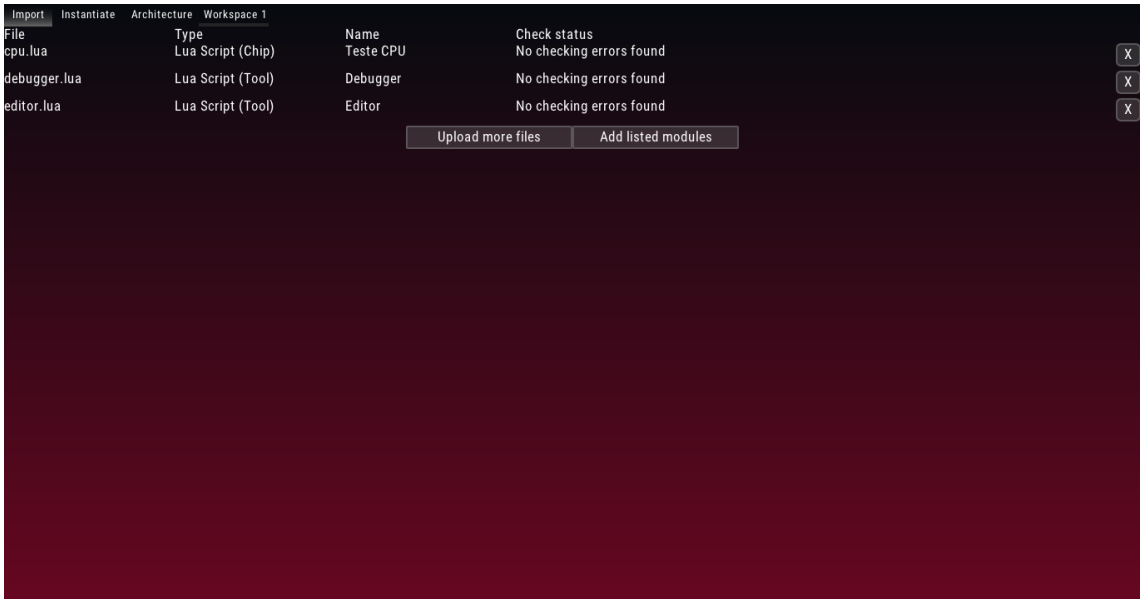

Figura 3. Importando modulos para a plataforma

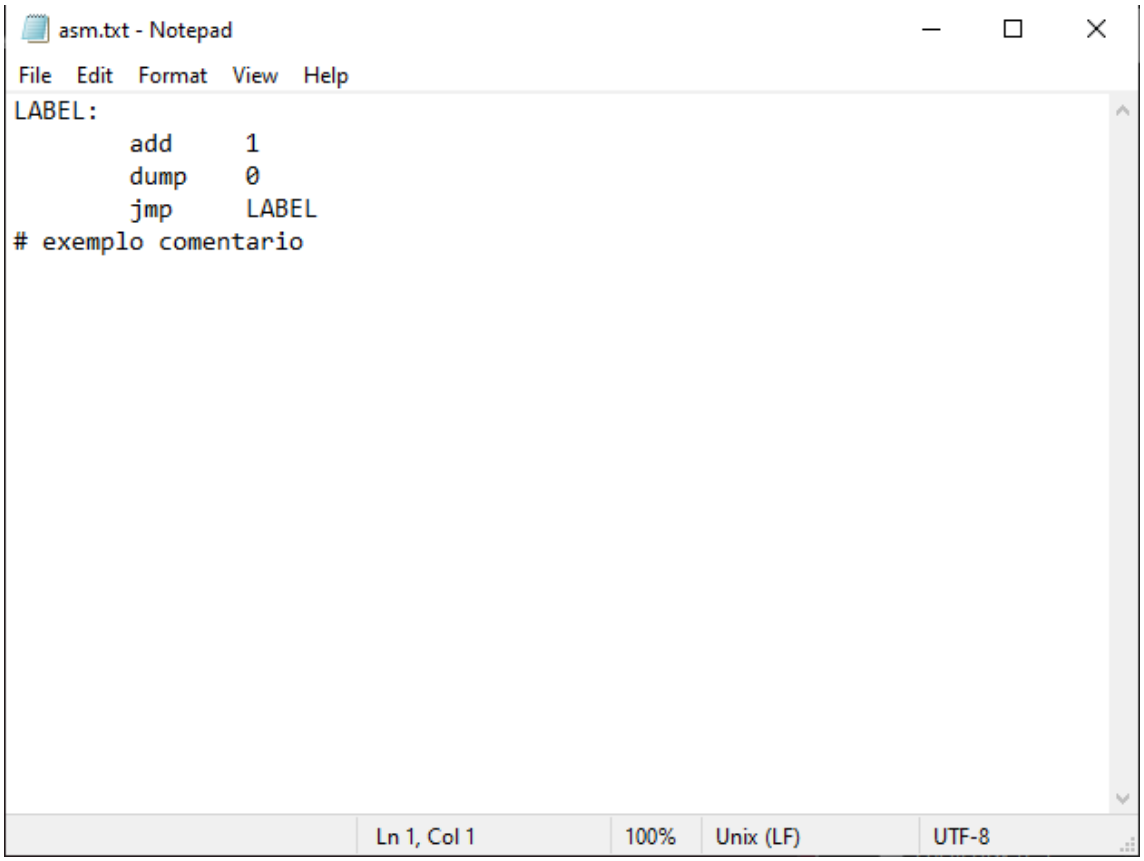

Figura 4. Exemplo do código em linguagem de montagem exportado para um arquivo de texto

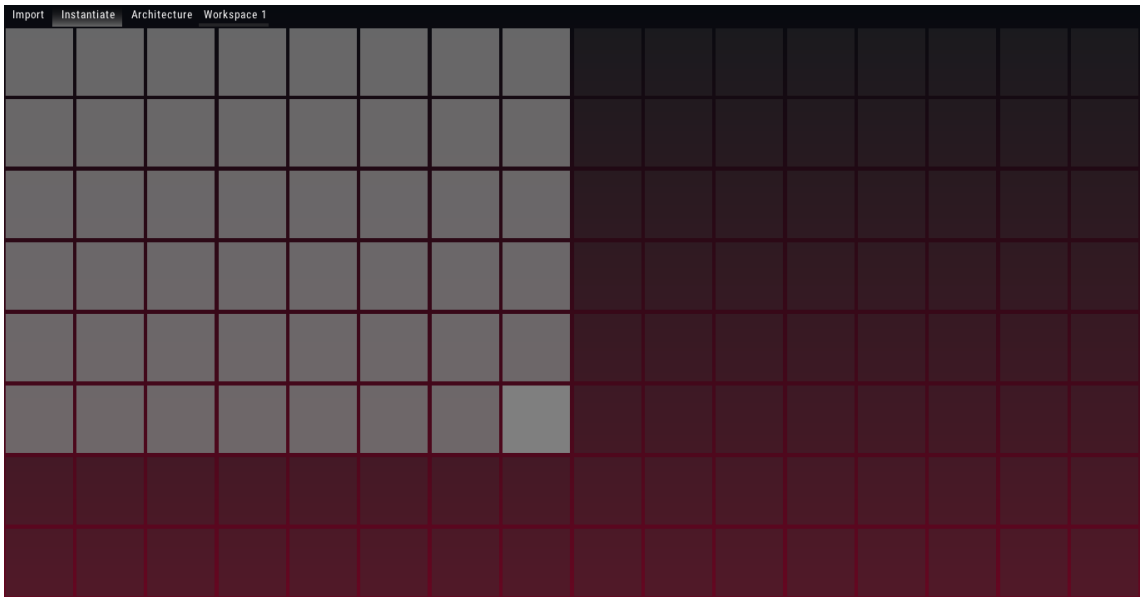

Figura 5. Instanciando um módulo

5. Conclusão

O objetivo foi atingido parcialmente: foi possível criar uma aplicação capaz de ser um ambiente frutífero para o aprendizado de programação de baixo nível mas que necessita de validação por meio de testes com alunos reais. Uma plataforma onde o aluno pode interagir com flexibilidade a arquiteturas virtuais definidas por um tutor ou por ele mesmo. As possibilidades para os módulos visuais são praticamente infinitas: visualizadores de registradores, debuggers, visualizadores de VRAM, visualizadores RAM, editores, entre outros. Se o tutor quiser explorar alguma maneira diferente de visualizar os dados de uma arquitetura virtual, o mesmo é capaz. A mesma coisa para a arquitetura virtual, o tutor pode definir desde arquiteturas RISC, simular arquiteturas antigas, até mesmo pensar e elaborar arquiteturas conceituais e pouco utilizadas.

Como principais contribuições deste trabalho, é possível evidenciar contribuições metodológicas como uma nova maneira de se pensar softwares de aprendizado de programação de baixo nível, especialmente no tocante ao sistema módular focado na interação tutor aluno (e de como o aluno pode transicionar gradualmente para o papel de tutor), e no sistema de workspaces. Como contribuições técnicas, o código fonte ficará disponibilizado na plataforma GitHub. E como contribuição social, a meta é que a versão futura do software possa estar pronta para ser usada ativamente por alunos da universidade.

Para trabalhos futuros: aplicar testes rigorosos para medir a qualidade do aprendizado, usabilidade e estabilidade da aplicação em usuários reais. Aplicar as modificações no software para aquelas características evidenciadas como não ideais prescritas na seção de resultados. Melhorar a qualidade da codebase. Buscar implementar novos módulos, tanto para testar melhor o sistema de módulos, quanto para desenvolver uma biblioteca pública de módulos para que novos tutores, ou alunos sem tutores disponíveis possam engrenar no processo de aprendizagem.

Referências

[Bolanakis et al. 2011] Bolanakis, D. E., Evangelakis, G. A., Glavas, E., and Kotsis, K. T. (2011). A teaching approach for bridging the gap between low-level and high-level programming using assembly language learning for small microcontrollers. *Computer applications in engineering education*, 19(3):525–537.

[da Silva et al. 2018] da Silva, T. S. C., de Melo, J. C. B., and Tedesco, P. C. d. A. R. (2018). Um modelo para promover o engajamento estudantil no aprendizado de programação utilizando gamification. *Revista brasileira de informática na educação*, 26(3):120.

[Eben Upton 2016] Eben Upton, Jeffrey Duntemann, R. R. T. M. B. E. (2016). *Learning Computer Architecture with Raspberry Pi*. Wiley, New York.

[Gallego-Durán et al. 2021] Gallego-Durán, F. J., Satorre-Cuerda, R., Compañ-Rosique, P., Villagrá-Arnedo, C. J., Molina-Carmona, R., and Llorens-Largo, F. (2021). A low-level approach to improve programming learning. *Universal access in the information society*, 20(3):479–493.

[Group 2023] Group, M. (2023). The talent shortage. https://go.manpowergroup.com/talent-shortage.

[Katagai and Fukuda 2017] Katagai, J. and Fukuda, H. (2017). Low level programming learning support system by implementation of virtual machine. *Proceedings of Annual Conference of the Information Systems Society in Japan*, page c24.

[Mack 2011] Mack, C. A. (2011). Fifty years of moore's law. *IEEE transactions on semiconductor manufacturing*, 24(2):202–207.

[Overflow 2022] Overflow, S. (2022). Survey 2022. https://survey.stackoverflow.co/2022.

[Pegoraro and Franchin 2022] Pegoraro, R. and Franchin, M. N. (2022). Um simulador de apoio ao ensino de linguagem de montagem. *Revista brasileira de ensino de ciência e tecnologia*, 15(2).

[Rahim et al. 2022] Rahim, S., Omar, S., Au, T. W., and Mashud, I. M. (2022). Smac-based programming learning tool: Validating a novel system architecture. *International journal of emerging technologies in learning*, 17(13):101–118.