

# **Relatório Trabalho Prático - Montador RISC-V**

**Erian Alves<sup>1</sup>, Guilherme Sérgio<sup>2</sup>, Pedro Cardoso<sup>3</sup>**

<sup>1</sup>Instituto de Ciências Exatas e Tecnológicas,  
Universidade Federal de Viçosa, Florestal, MG, Brasil

## **1. Introdução**

RISC-V é um conjunto de instruções (ISA) baseado e estabelecido nos princípios RISC. Ele é livre para ser usado para qualquer finalidade, permitindo a qualquer pessoa ou empresa projetar e vender chips e software RISC-V. O projeto começou em 2010 na Universidade da Califórnia, em Berkeley.

O RISC-V foi projetado para implementações de alto desempenho e baixo consumo de energia. Sendo um conjunto limpo e modular, trabalhando com bases de 32, 64 e 128 bits, com várias opções de extensão em ponto flutuante. O objetivo dos autores do RISC-V era de fornecer diversos designs de CPU diferentes sob a licença BSD. Tal licença permite que designs baseados em RISC-V possam ser implementados tanto de maneira aberta como proprietária, ao contrário de outros designs como ARM ou MIPS, que cobram taxas pelo uso de suas patentes além de requererem acordos de não-divulgação para liberação de suas respectivas documentações. O seu caráter aberto favorece a sua utilização para fins educacionais.

Visto isso, o objetivo deste trabalho é implementar uma versão simplificada de um montador RISC-V, o qual deve ser compatível com a codificação do livro texto, podendo ser utilizada qualquer linguagem de programação na implementação.

## **2. Desenvolvimento**

Para a produção deste trabalho foi escolhida a linguagem de programação Python devido à sua dinamicidade no tratamento de strings, a qual era fundamental para a tarefa solicitada.

A atividade proposta consistia na implementação de um montador RISC-V que suportasse um determinado conjunto de instruções, o qual estava dividido entre os tipos R e I. O formato das instruções é composto do nome da operação e os operandos utilizados, como mostra o exemplo:

`“nomeOperacao” “operando1”, “operando2”, “operando3”`

No arquivo de entrada do programa implementado, cada linha corresponde ao molde apresentado.

### **2.1. Implementação do montador**

A fim de transcrever o código em assembly para o seu correspondente em linguagem de máquina, funções foram criadas e atribuídas a esse processo. Dentre as tarefas realizadas por elas estavam a mudança de base numérica de valores na entrada e a associação das informações de uma instrução a partir do nome. Para promover uma generalização desse processo de identificação, utilizou-se de uma estrutura de dados nativa da linguagem escolhida. O dicionário, conhecido como vetores associativos em outras linguagens, em

Python, trata-se de uma coleção não ordenada, mutável e indexável. É organizada da seguinte forma:

```
dados = {  
    "instru1" : "info1",  
    "instru2" : "info2",  
    "instru3" : "info3"  
}
```

Ademais, outras ferramentas empregadas foram as funções embutidas, como `abs()` e `bin()`, as quais retornam o valor absoluto de um número e a versão binária de um inteiro, respectivamente, por exemplo.

O funcionamento do programa fundamenta-se em analisar uma linha por vez do arquivo de entrada, sendo o processo de conversão bastante semelhante para todos os tipos implementados. O trecho o qual corresponde ao nome da instrução é averiguado e identifica-se, a partir de uma lista, a qual categoria ele pertence, para que seja chamada a função apropriada. Em seguida, como apresentado anteriormente, é recuperado dentro de um dicionário as informações relacionadas aos campos `funct3`, `funct7` ou `immediate`, dependendo do tipo em questão, a partir do nome do processo. Na sequência, os outros blocos da instrução são analisados, sejam registradores ou inteiros, e é chamada a função de conversão de base para obter seus valores binários. Por fim, é realizado alguns ajustes para deixar os números com a quantidade de bits no padrão necessário e cada parte das informações é organizada em uma string, na ordem correta, e esse resultado é retornado para o usuário.

Com o intuito de promover uma melhor organização, os procedimentos foram distribuídos em vários arquivos, facilitando na identificação de erros e eventuais alterações no código. Além disso, também foi estabelecido um mecanismo, por meio de métodos já citados, de identificação se o resultado deve ser mostrado no terminal ou escrito em um arquivo de saída.

Em relação a implementação de recursos extras, o montador é capaz de reconhecer a base numérica hexadecimal no código assembly, além do padrão decimal. Ademais, houve o acréscimo de instruções de desvio para a checagem de casos nos quais os valores sejam iguais (`beq`), caso não sejam (`bnq`), em uma situação que o número é menor que o outro (`blt`) e o oposto, maior que ou igual (`bge`). Além da inserção das operações de `load` e `store`. E por fim, a incorporação, também, de algumas pseudo-instruções citadas no livro texto, como a de carregar um número em um registrador, copiar um valor de um registrador para outro e a capacidade de reconhecer que uma instrução `"and"` com um inteiro corresponde a um `"andi"`. Esses aparatos tem como propósito simplificar o serviço dos desenvolvedores trabalhando com o assembly do RISC-V.

## 2.2. Executando o programa

Como já dito, o trabalho foi feito utilizando a linguagem python, que é uma linguagem interpretada. Assim, para rodar o programa é necessário ter o interpretador python instalado na máquina. É importante salientar também que a versão instalada deve ser 3.x. Em distribuições Linux, o python geralmente vem instalado por padrão. É possível verificar se

o python já está instalado usando o comando “Which python”, e caso ele esteja instalado, é possível verificar sua versão utilizando o comando “python –version” no terminal.

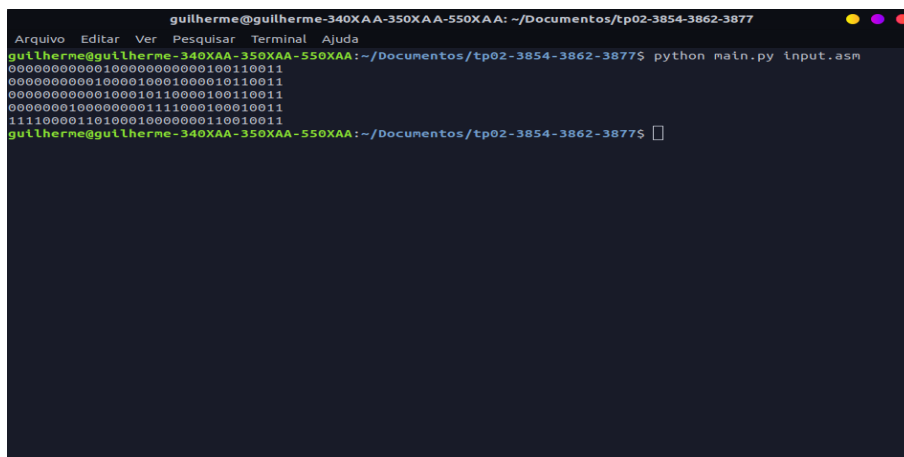
Na hipótese de o python não estar instalado ou de ser uma versão inferior, basta utilizar o comando “sudo apt-get install python3” no terminal, assim o python 3 será instalado na máquina.

Após certificado que a instalação do python está concluída, o programa pode ser executado. O código pode ser rodado de duas formas, passando um arquivo de saída ou não. Caso o usuário não passe o arquivo de saída, o output é imprimido no terminal.

Para executar o programa e escrever o resultado em um arquivo de saída basta executar o comando:

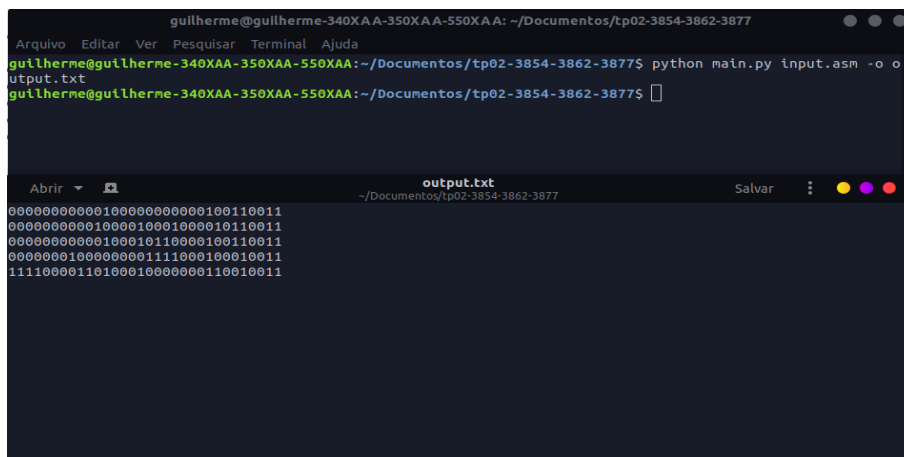
*python main.py entrada.asm -o saida*

Para executar o programa e imprimir a saída no terminal basta executar o comando acima sem a opção de saída (-o saida).

A terminal window with a dark background. The title bar shows the user 'guilherme' and the path '~/Documentos/tp02-3854-3862-3877'. The terminal content shows the command 'python main.py input.asm' being executed. The output consists of seven lines of binary code (0s and 1s) printed in green text. The prompt 'guilherme@guilherme-340XAA-350XAA-550XAA:~/Documentos/tp02-3854-3862-3877\$' is visible at the bottom.


**Figura 1. Exemplo de saída no terminal**

A figura acima mostra um exemplo de saída no terminal, utilizando o arquivo de entrada genérico.

Two screenshots. The top one is a terminal window showing the command 'python main.py input.asm -o output.txt' being executed. The bottom one is a file editor window titled 'output.txt' showing the same seven lines of binary code that were printed in the terminal above. The file editor has a menu bar with 'Abrir', 'Salvar', and other options.

**Figura 2. Exemplo de saída em um arquivo**

A figura 2 é um exemplo de saída com arquivo, onde mostra o terminal e o arquivo de saída sobrepostos. É possível notar que é preciso passar a extensão do arquivo de saída, como no exemplo, que foi .txt.

A screenshot of a text editor window with a dark background. The title bar at the top shows 'input.asm' and a file path '~/.Documentos/tp02-3854-3862-3877'. The window contains five lines of assembly code: 'add x2, x0, x1', 'sll x1, x2, x2', 'or x2, x2, x1', 'andi x2, x1, 16', and 'addi x3, x2, -243'. The interface includes a menu bar with 'Abrir' and 'Salvar' options, and standard window control buttons (minimize, maximize, close) on the right.

**Figura 3. Entrada utilizada nos exemplos**

A figura 3 contém o arquivo .asm que foi utilizado como entrada para executar os exemplos das figuras 1 e 2.

### 3. Considerações finais

A partir da produção deste trabalho, os assuntos relacionados ao conjunto de instruções RISC-V, o qual foi estudado em aula, foram colocados em prática proporcionando um entendimento maior sobre o tema. Foi possível ver mais de perto os princípios do RISC-V para um bom design, como o compromisso entre grandes constantes/endereços e tamanhos de instruções similares. Além disso, como o montador é uma ferramenta que compõe o processo de traduzir um programa em linguagem de alto nível até o ponto de ser executado pelo hardware, a sua implementação promoveu reflexões acerca do funcionamento do procedimento e contribuiu para uma melhor compreensão desse.

### 4. Referências

- [1] RISC-V. Wikipédia. Disponível em: <<https://pt.wikipedia.org/wiki/RISC-V>>. Acesso em: 17 de out. de 2020.
- [2] Python Dictionaries. W3Schools. Disponível em: <[https://www.w3schools.com/python/python\\_dictionaries.asp](https://www.w3schools.com/python/python_dictionaries.asp)>. Acesso em: 19 de out. de 2020.
- [3] Data Structures. Python. Disponível em: <<https://docs.python.org/3/tutorial/datastructures.html#dictionaries>>. Acesso em: 19 de out. de 2020.
- [4] David A. Patterson, John L. Hennessy. Computer Organization and Design RISC-V Edition: The Hardware Software Interface. Morgan Kaufmann. 1ª edição, 2017. Acesso em: out. de 2020.