

Montador Simplificado para Instruções RISC-V

Gabriel S. Avelino¹, Kaua Kirk²

¹Universidade Federal de Ouro Preto (UFOP)
Caixa Postal 24 - CEP 35.931-008 - João Monlevade - MG - Brasil

²Departamento de Computação e Sistemas
Universidade Federal de Ouro Preto (UFOP) – João Monlevade, MG – Brasil

Abstract. *Este trabalho descreve a implementação de um montador RISC-V simplificado, desenvolvido como parte da disciplina CSI211 – Fundamentos de Organização e Arquitetura de Computadores. O montador traduz instruções do formato assembly para a codificação binária de 32 bits. O projeto foi implementado em Python com suporte a instruções básicas e oferece funcionalidade de saída em terminal ou arquivo.*

1. Introdução

O presente trabalho tem como objetivo a implementação de um montador para um subconjunto das instruções da arquitetura RISC-V. O montador traduz instruções escritas em linguagem de montagem para a representação binária de 32 bits conforme as especificações do padrão.

2. Instruções Suportadas

As instruções suportadas são (dupla 13):

- lb, sb
- add, and
- ori, sll
- bne

3. Requisitos

O montador foi desenvolvido em *Python* e para que seja executado é necessário:

- Python 3.8 ou superior
- Sistema operacional: Windows ou Linux

4. Instalação do Python

4.1. Linux

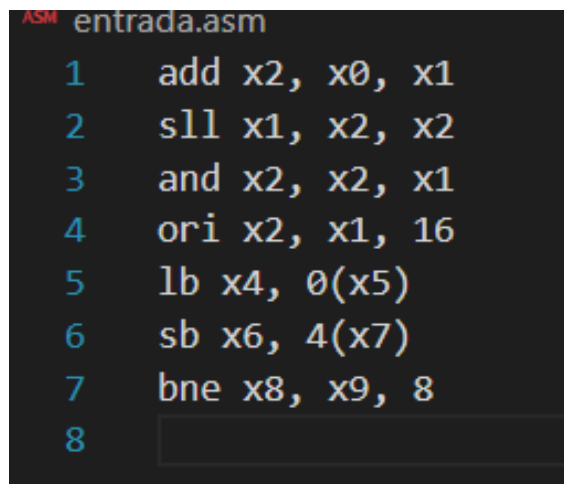
- Atualize os repositórios: `sudo apt update`
- Instale o Python: `sudo apt install python3 python3-pip -y`
- Verifique a instalação: `python3 --version`

4.2. Windows

- Acesse: <https://www.python.org/downloads/>
- Faça o download da versão mais recente
- Marque "Add Python to PATH" durante a instalação
- Verifique: `python --version`

5. Código Utilizado

- add x2, x0, x1
- sll x1, x2, x2
- and x2, x2, x1
- ori x2, x1, 16
- lb x4, 0(x5)
- sb x6, 4(x7)
- bne x8, x9, 8

A screenshot of a code editor with a dark background. The title bar at the top left says 'ASM' in red and 'entrada.asm' in white. The code is written in a light blue font and is numbered from 1 to 8 on the left side. The code consists of seven instructions: 'add x2, x0, x1', 'sll x1, x2, x2', 'and x2, x2, x1', 'ori x2, x1, 16', 'lb x4, 0(x5)', 'sb x6, 4(x7)', and 'bne x8, x9, 8'. The eighth line is empty and has a text input field.

```
ASM entrada.asm
1  add x2, x0, x1
2  sll x1, x2, x2
3  and x2, x2, x1
4  ori x2, x1, 16
5  lb x4, 0(x5)
6  sb x6, 4(x7)
7  bne x8, x9, 8
8  
```

Figure 1. Instruções utilizadas nos testes

6. Como Executar o Montador

6.1. Saída em binário

```
python montador.py entrada.asm -o saida.txt
python montador.py entrada.asm
```

6.2. Saída em hexadecimal

```
python montador.py entrada.asm -o saida.txt --hex
```

7. Funcionamento Interno

O funcionamento do montador envolve várias etapas organizadas de forma eficiente. Abaixo estão os principais pontos:

7.1. Leitura do Arquivo

```
with open(entrada, 'r') as f:
    linhas = f.readlines()
```

O conteúdo do arquivo '.asm' é lido linha a linha, armazenando todas em uma lista para posterior processamento.

```

# saída.txt
1 00000000000100000000000100110011
2 00000000001000010001000010110011
3 00000000000100010111000100110011
4 000000100000000111000100010011
5 000000000000010100001000000011
6 0000000001100011100001000100011
7 00000000100101000001010001100011
8

PROBLEMAS SAÍDA CONSOLE DE DEPURACÃO TERMINAL PORTAS

python montador.py entrada.asm

00000000000100000000000100110011
00000000001000010001000010110011
00000000000100010111000100110011
000000010000000011000100010011
000000000000010100001000000011
0000000001100011100001000100011
00000000100101000001010001100011
00000000100101000001010001100011

```

Figure 2. Saída em binário no terminal e no arquivo

```

# saída.txt
1 100133
2 2110b3
3 117133
4 100e113
5 28203
6 638223
7 941463
8

PROBLEMAS SAÍDA CONSOLE DE DEPURACÃO TERMINAL PORTAS

PS C:\Users\gabriel\OneDrive\Área de Trabalho\Tpoac> python montador.py entrada.asm -o saída.txt --hex
Montagem concluída! Arquivo salvo em: saída.txt
PS C:\Users\gabriel\OneDrive\Área de Trabalho\Tpoac>

```

Figure 3. Saída em hexadecimal

7.2. Função `traduzir_linha(linha)`

Essa função é responsável por converter uma única linha de código assembly em sua representação binária de 32 bits.

- Entrada: string de instrução (ex: `add x1, x2, x3`)
- Saída: string binária (ex: `000000000011000100000000110110011`)

7.3. Conversão para Hexadecimal

Caso a flag `--hex` seja utilizada, cada instrução binária é convertida com:

```
hex(int(binario, 2))[2:]
```

O método remove o prefixo `0x` e retorna a representação hexadecimal pura.

7.4. Execução Condicional

O programa identifica a flag opcional `--hex` e decide como gerar a saída:

```
if hexadecimal:
    f.write(f"{hex(int(b, 2))[2:]}\n")
else:
    f.write(b + '\n')
```

8. Tratamento de Erros

8.1. Instruções Inválidas

Cada linha é processada com um bloco de exceção:

```
try:
    binario = traduzir_linha(linha)
except Exception as e:
    print(f"Erro na linha: {linha}")
    sys.exit(1)
```

8.2. Arquivo Não Encontrado

Se o arquivo de entrada não existir:

```
except FileNotFoundError:
    print(f"Arquivo não encontrado: {entrada}")
    sys.exit(1)
```

Esses blocos evitam falhas silenciosas e ajudam no diagnóstico rápido. y

9. Conclusão

O montador desenvolvido permite converter instruções RISC-V de um arquivo ‘.asm’ para binário ou hexadecimal, com uso opcional de argumentos. O projeto mostrou como é possível aplicar conceitos de arquitetura de computadores usando uma linguagem moderna como Python. O uso de tratamento de erros e flags de execução torna o programa mais robusto. Futuramente, pretende-se adicionar mais instruções, suporte a pseudo-instruções e melhorias no reconhecimento de sintaxe.

10. Acesso ao Projeto no Overleaf

O projeto completo pode ser acessado em:

[Clique aqui para abrir no Overleaf](#)